Report No. UIUCDCS-R-76-787

NSF-OCA-DCR73-07980 A02-000018

THE STATISTICAL THEORY OF RELATIVE ERRORS IN FLOATING-POINT COMPUTATION

by

Douglass Stott Parker, Jr.

March 1976

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-76-787

THE STATISTICAL THEORY OF RELATIVE ERRORS IN FLOATING-POINT COMPUTATION

by

Douglass Stott Parker, Jr.

March 1976

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under Grant No. US NSF DCR73-07980 A02 and was submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, March 1976.

TABLE OF CONTENTS

1. INTRODUCTION

For the man on the street, roundoff analysis of floating-point computation in synonymous with keeping bounds on accumulated errors: at any stage in some computation, the computed approximation fl(A) to a partial result A satisfies

$$fl(A) = A(1 + \delta)$$

where $\delta = (fl(A) - A)/A$ is the relative error of the computation [16]. The idea is to find limits on the magnitude of δ.

Recently, however, a number of papers have appeared adding probabilistic considerations to the distribution of roundoff errors in single arithmetic operations and of--what almost amounts to the same--errors in representing the reals as floating-point numbers ([3], [6], [14]). They show that relative errors are not uniformly distributed within their bounds, but instead cluster around zero. The error bounds themselves are very pessimistic.

In this paper relative errors δ are viewed as random variables with suitable probability distributions. The results of applying classical probability theory to roundoff analysis in this fashion are: (1) A proof of the generally accepted statement that accumulated errors approach normal distributions after several operations ([13, p. 113], [11, p. 104], [4, p. 306]). (2) An appreciation of the rate of convergence to normal distributions and how floating-point addition and correlated errors can slow this convergence down. (3) A new method of

automated error analysis comparable to (but in most ways inferior to) Interval Analysis, but in every way superior to Wilkinson bound analysis.

We should comment that it is not "unrealistic" to treat relative errors as random variables. To do so is just to follow the basic philosophy of Bayesian statistics: if you don't know the exact value of a variable, but know what its possible values are and what the probability of taking each of those values on is, then you should treat the variable as random with the corresponding probability distribution. Treating roundoff errors in this way makes our work fairly clean, plus lets it be general and rigorous as well.

2. RELATIVE ERRORS AS PROBABILITY DENSITIES

In the analysis of Wilkinson [16] the relative error $\delta_A = (fl(A) - A)/A$ in a computed result A is rarely known exactly--usually one says that it lies within known bounds. These bounds are proportional to the machine precision β^{-t}, where β is the arithmetic base and t is the number of mantissa digits used in representing floating-point numbers. Computation of bounds is simple: if the floating-point approximations to A and B involve the relative errors δ_A and δ_B, respectively, then the floating-point approximation to $A \cdot B$ is

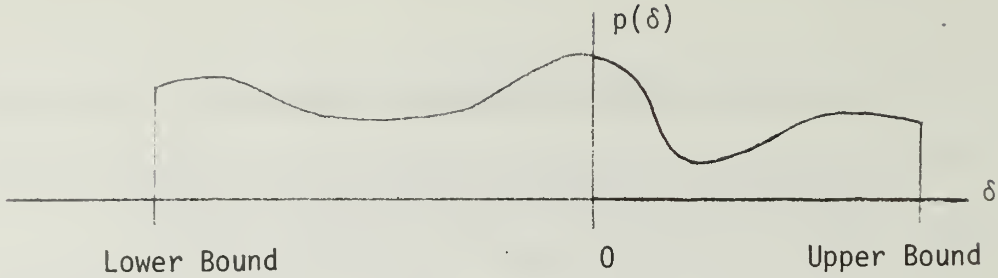
$$\begin{aligned} fl(A \cdot B) &= [A(1 + \delta_A)][B(1 + \delta_B)](1 + \delta_R) \\ &= AB(1 + \delta_A + \delta_B + \delta_R + \delta_A\delta_B + \delta_A\delta_R \\ &\quad + \delta_B\delta_R + \delta_A\delta_B\delta_R) \end{aligned} \tag{2.1}$$

where δ_R is the rounding error incurred in the multiplication and is bounded by the machine precision. Since hopefully $|\delta_A| << 1$, $|\delta_B| << 1$, we have

$$fl(AB) \approx AB(1 + \delta_A + \delta_B + \delta_R) \tag{2.2}$$

to a very good approximation. Then the bound for the error on fl(AB) is the sum of the bounds for δ_A, δ_B, and δ_R.

Up to this point we have treated each relative error δ as an unknown element of a known interval. It is not a big step to view δ as being smeared over this interval, having at each point a probability of occurrence:



Thus we equate δ with an underlying probability density $p(\delta)$ so that

$$\text{Probability } (\delta \leqq x_0) = \int_{-\infty}^{x_0} p(x)\, dx$$

$$= \int_{\text{Lower Bound}}^{x_0} p(x)\, dx \qquad (2.3)$$

Recall that a <u>probability density</u> f is a function on $\mathbb{R}$ satisfying

$$\text{(i)} \quad f(x) \geqq 0 \quad \forall\, x \,\varepsilon \mathbb{R}$$

$$\text{(ii)} \quad \int_{-\infty}^{\infty} f(x)\, dx = 1 \qquad (2.4)$$

and that the <u>probability distribution</u> F corresponding to the density f is defined by

$$F(x) = \int_{-\infty}^{x} f(t)\, dt \qquad (2.5)$$

The question arises as to what expressions like "$A(1 + \delta_A)$" mean if we view δ_A as a probability density. This is resolved by the following definition and explanation.

<u>Def</u>: A <u>smeared number</u> (or <u>fuzzy number</u>) is a pair $N = [A,f]$ where $A \varepsilon \mathbb{R}$ and f is a probability density on $\mathbb{R}$. The probability that N takes on a value $\leqq A \cdot C$ for any $C \varepsilon \mathbb{R}$ is given by

$$Pr(N \leqq A \cdot C) = \int_{-\infty}^{C} f(t)\, dt \qquad (2.6)$$

The idea is that <u>every floating-point approximation to a number is a smeared number</u>. Thus $fl(A) = A(1 + \delta_A) = [A,(1 + \delta_A)]$. We clarify this concept with three observations:

Observation 1. The constant 1 in expressions of the form $(1 + \delta)$ should be thought of as an impulse function Δ (Dirac delta) at 1.

This is natural since 1 should represent a density with all its support at 1. In the following we will use 1 and Δ interchangeably when no confusion should result.

Observation 2. If δ_1 and δ_2 are independent random variables with corresponding densities p_1 and p_2, then $\delta = \delta_1 + \delta_2$ corresponds to the convolution density $p = p_1 * p_2$.

Since δ is the density which gives the probability at each point x that $\delta_1 + \delta_2 = x$,

$$Pr(\delta \leqq x) = \iint_{y + z \leqq x} p_1(y)\, p_2(z)\, dydz$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{x - z} p_1(y)\, p_2(z)\, dydz$$

Differentiating with respect to x,

$$p(x) = \int_{-\infty}^{\infty} p_1(x - z)\, p_2(z)\, dz = p_1 * p_2(x)$$

Note from observations 1 and 2 that if p_A is the density corresponding to δ_A, then

$$fl(A) = A(1 + \delta_A) = [A, (1 + \delta_A)] = [A, \Delta \star p_A]$$

Observation 3. If δ_1 and δ_2 are independent random variables with corresponding densities p_1 and p_2, then $\delta = \delta_1 \delta_2$ corresponds to the density p defined by

$$p(x) = \int_{-\infty}^{\infty} p_1(x/z)\, p_2(z)\, \frac{dz}{z}.$$

The derivation is similar to that for Observation 2. We remark that the approximation

$$(1 + \delta_1 + \delta_2 + \delta_1 \delta_2) \approx (1 + \delta_1 + \delta_2)$$

when δ_1 and δ_2 are small random variables (with corresponding densities) still makes sense: if p_1 is zero outside $[-D_1, D_1]$ and p_2 is zero outside $[-D_2, D_2]$ then the density p defined as above for $\delta_1 \delta_2$ is zero outside $[-D_1 D_2, D_1 D_2]$. Assuming $D_1, D_2 << 1$, p approximates an impulse function at zero (the identity for convolution).

We now consider what fl(A + B) looks like. Since

$$fl(A + B) = (A(1 + \delta_A) + B(1 + \delta_A))\ (1 + \delta_R)$$

$$= A(1 + \delta_A + \delta_R) + B(1 + \delta_A + \delta_R)$$

where again δ_R is rounding error, we start with

$$fl(A + B) \approx [A, 1 \star p_A \star p_R] + [B, 1 \star p_B \star p_R]$$

The results for subtraction and division are similar. We summarize the results here:

$$fl(A + B) = [A, 1 * p_A * p_R] + [B, 1 * p_B * p_R] \tag{2.7}$$

$$fl(A - B) = [A, 1 * p_A * p_R] + [-B, 1 * p_B * p_R] \tag{2.8}$$

$$fl(A \cdot B) = [A \cdot B, 1 * p_A * p_B * p_R] \tag{2.9}$$

$$fl(A/B) = [A/B, 1 * p_A * \check{p}_B * p_R] \tag{2.10}$$

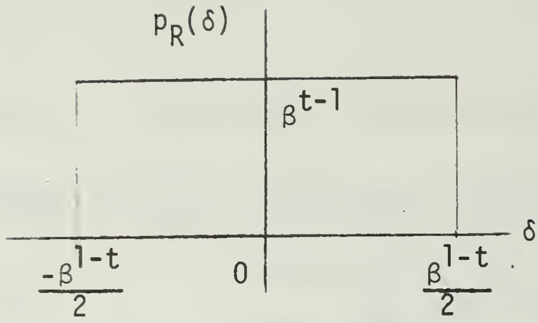
In (2.10), $\check{p}_B$ is defined by $\check{p}_B(x) = p_B(-x)$. Notice that in (2.7) and (2.8) the result is not actually a smeared number but a sum of smeared numbers. This reflects Wilkinson's statement [16, §28] that you cannot say much about the error in a sum without knowing more about the magnitudes of the summands. In the next section we show a method to reduce the sum to a single smeared number. In short, we show that we can make a consistent algebra for smeared numbers.

We must now select probability densities $p(\delta)$ for (1) representation errors (i.e., errors incurred by representing real numbers in floating-point format) and (2) the rounding error δ_R made in single arithmetic operations. All other errors are sums of these two kinds. As mentioned in the introduction, we can use the same density for both (1) and (2):

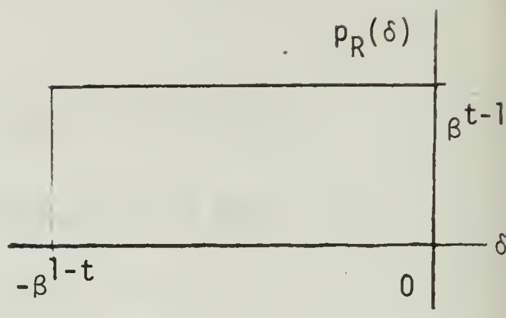
Let p_R be the uniform probability density on

$[-\beta^{1-t}/2, \beta^{1-t}/2]$ if <u>rounded</u> arithmetic is used (2.11)

$[-\beta^{1-t}, 0]$ if <u>chopped</u> (truncated) arithmetic is used



p_R for Rounded Arithmetic



p_R for Chopped Arithmetic

Note that p_R is a uniform distribution inside the ordinary Wilkinsonian bounds. This is very pessimistic but is the best one can do a priori. If we consider the error in representing $A = \alpha \cdot \beta^e$ where $\alpha \epsilon [1/\beta, 1)$, $e \epsilon \mathbb{Z}$ then since

$$[fl(\alpha) - \alpha] \tag{2.12}$$

is uniformly distributed on $[-\beta^{-t}/2, \beta^{-t}/2]$ for rounding or $[-\beta^{-t}, 0]$ for chopping (because the "distribution of (t + 1)-th digits" is practically uniform; cf. [14, p. 270])

$$\delta_A = (fl(A) - A)/A = (fl(\alpha) - \alpha)/\alpha \tag{2.13}$$

is uniformly distributed on $[-\beta^{-t}/2\alpha, \beta^{-t}/2\alpha]$ for rounding or $[-\beta^{-t}/\alpha, 0]$ for chopping. If we assume the worst case $\alpha = 1/\beta$, we get the p_R defined above.

3. WHY ACCUMULATED ERROR DENSITIES ARE ALMOST NORMAL

This section introduces the Central Limit Theorem, which is the basis for the popular statement that accumulated relative errors become normally distributed. The Central Limit Theorem is useful intuitively but we avoid its proof, offering Section 4 instead. In this section we explain how a sum of relative errors converges to a normally distributed error. After showing that sums of smeared numbers are smeared numbers we get an appreciation of how this convergence may be disturbed.

We begin by reviewing some important concepts from probability theory. Let ξ be a random variable having corresponding probability density, f. Then:

Def: The nth moment M_n of ξ (or of f) is

$$M_n = \int_{-\infty}^{\infty} x^n f(x)\, dx. \tag{3.1}$$

Def: Let g be an arbitrary complex-valued function on $\mathbb{R}$. Define the expected value of $g(\xi)$ by

$$E[g(\xi)] = \int_{-\infty}^{\infty} g(x) f(x)\, dx \tag{3.2}$$

Note therefore that $M_n = E[\xi^n]$.

Def: The mean of ξ (or of f) is its first moment:

$$\mu = E[\xi] = M_1 \tag{3.3}$$

<u>Def</u>: The <u>variance</u> of ξ (or of f) is given by

$$\sigma^2 = E[(\xi - \mu)^2] = M_2 - M_1^2 \tag{3.4}$$

The definitions (3.1) - (3.4) are tied together with the following theorem:

<u>Theorem 1</u>. Given densities $f_1, \ldots, f_N$ where f_i has mean μ_i and variance σ_i^2, then the convolution density $f_1 * \ldots * f_N$ has mean $\mu = \sum_{i=1}^{N} \mu_i$ and variance $\sigma^2 = \sum_{i=1}^{N} \sigma_i^2$.

<u>Proof</u>. Compute the Fourier transforms ("characteristic functions," $E[\exp(iw\xi)]$) of both sides of $f = f_1 * \ldots * f_N$ and use the facts that

$$(1) \quad \hat{g}(w) = \sum_{n=0}^{\infty} \frac{(-i)^n M_n}{n!} \qquad (i = \sqrt{-1} \; ; \; \hat{} = \text{Fourier Transform})$$

$$(2) \quad \widehat{g * h}(w) = \hat{g}(w)\, \hat{h}(w)$$

Now recall that the Normal (or Gaussian) probability density with parameters $\mu, \sigma (\sigma > 0)$ is

$$\psi(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-(x-\mu)^2/2\sigma^2} \tag{3.5}$$

so its probability distribution $\Phi(x)$ is

$$\Phi(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \int_{-\infty}^{x} e^{-(t-\mu)^2/2\sigma^2}\, dt \tag{3.6}$$

It is easy to show that the mean and variance of ψ are μ and σ^2, respectively.

The Central Limit Theorem may be stated as follows: given a set $\{\delta_1, \ldots, \delta_N\}$ of random variables having respective probability densities p_i, means μ_i, and variances σ_i^2 $(i = 1, \ldots, N)$ then under "certain loose conditions" the sum variable $\delta = \delta_1 + \delta_2 + \ldots + \delta_N$ (having density $p = p_1 * p_2 * \ldots * p_N$) converges as $N \to \infty$ to a normal density with mean $\mu = \mu_1 + \mu_2 + \ldots + \mu_N$ and variance $\sigma^2 = \sigma_1^2 + \sigma_2^2 + \ldots + \sigma_N^2$.

Proof of the Central Limit Theorem is dependent of course on the "loose conditions" surrounding the summand variables and the type of convergence used. Good surveys may be found in [1] and [12]. We will omit any proof here for roundoff errors because the convergence to a normal distribution, although useful intuitively, is difficult to prove and unnecessary for obtaining statistical bounds (see Section 4).

The idea is that the smeared number $A(1 + \delta_1 + \delta_2 + \ldots \delta_N)$ is equal to $A(1 + \delta)$ where δ has a density that is almost normal. Since:

1. We can compute δ's mean μ and variance σ^2 easily.
2. Assuming δ normal, the probability that $|\delta - \mu| = 3\sigma$ is 0.002798 (i.e., a 99.7% confidence interval on δ's value is $[\mu - 3\sigma, \mu + 3\sigma]$).
3. For N larger than 2 or 3 the error bound $[\mu - 3\sigma, \mu + 3\sigma]$ is inside the crude Wilkinsonian error bound $[-N\beta^{1-t}/2, N\beta^{1-t}/2]$ for Rounding or $[-N\beta^{1-t}, 0]$ for truncation

--apparently without much effort we can get much tighter, more realistic error bounds and improve our error estimates.

This gives us our motivation: statistical error bounds seem tight. But we need more details.

We compute the mean and variance for the rounding error density p_R defined by (2.11). For <u>rounding</u>, we have

$$\mu_{Rounding} = 0$$

$$\sigma^2_{Rounding} = (\beta^{1-t}/2)^2/3 = \beta^{2-2t}/12 \qquad (3.7)$$

For <u>chopping</u>, similarly

$$\mu_{Chopping} = -\beta^{1-t}/2$$

$$\sigma^2_{Chopping} = \frac{(\beta^{1-t})^2}{3} - \frac{(-\beta^{1-t})^2}{2} = \beta^{2-2t}/12 \qquad (3.8)$$

Therefore

$$\sigma_R \overset{def}{\equiv} \sigma_{Rounding} = \sigma_{Chopping} = \beta^{1-t}/\sqrt{12} \qquad (3.9)$$

Notice that from (2.9) and (2.10) expressions involving only repeated multipliation, repeated division, or mixed multiplication and division will result in smeared numbers of the form $[A, 1 * \overbrace{p_R * p_R * p_R * \cdots p_R}^{m}]$ where the number m of p_R's in the convolution product depends of course on the number of operands in the expression and the accumulated error for each operand. For example if A,B,C are irrational then $fl(A(B\cdot C))$ $= [ABC, 1 * p_R * (p_R * p_R * p_R) * p_R]$ since each number has a representation error bounded by p_R, and the two multiplication operations have

rounding (truncation) errors bounded by p_R. The Wilkinson bounds for the relative error in ABC would be $\delta_{ABC} \in [-\frac{5}{2}\beta^{1-t}, \frac{5}{2}\beta^{1-t}]$ for rounding and $\delta_{ABC} \in [-5\beta^{1-t}, 0]$ for truncation.

If we assume (Central Limit Theorem) that $(p_R * p_R * p_R * p_R * p_R)$ is normal with $\sigma^2 = 5\sigma_R^2$, $\mu = 0$ for rounding or $\mu = 5(-1/2\beta^{1-t})$ for chopping, then the error in ABC should satisfy

$$\delta_{ABC} \in [\mu - 3\sigma, \mu + 3\sigma] = \begin{cases} [-\sqrt{\frac{5}{12}}\beta^{1-t}, 3\sqrt{\frac{5}{12}}\beta^{1-t}] & \text{Rounding} \\ [(\frac{-5}{12} - 3\sqrt{\frac{5}{12}})\beta^{1-t}, (\frac{-5}{2} + 3\sqrt{\frac{5}{12}})\beta^{1-t}] & \text{Chopping} \end{cases}$$

99.7% of the time. Now $3\sqrt{\frac{5}{12}} \approx 1.94 < 5/2$, and we have improved our error bounds by 20% after only two operations. As the number of operations N increases we will get better improvement since Wilkinson's bound are proportional to N while the 3σ bounds are proportional to $\sqrt{N}$.

It is not unreasonable to assume that $(p_R * p_R * p_R * p_R * p_R)$ is normal, either. Figures 1-4 show the result of repeated convolution with the error density for Rounding. Already $(p_R * p_R * p_R * p_R)$ is essentially normal (Figure 4). On a superficial basis at the least, convergence to a normal density seems rapid.

It should be pointed out that the use of division introduces "checked" densities $\check{p}_R$ in the convolution product (c.f. (2.10)). This is not a problem because

$$\check{p}_R(x) = p_R(-x) = \begin{cases} p_R(x) & \text{for Rounding} \\ p_R(x - \beta^{1-t}) & \text{for Chopping} \end{cases} \quad (3.10)$$

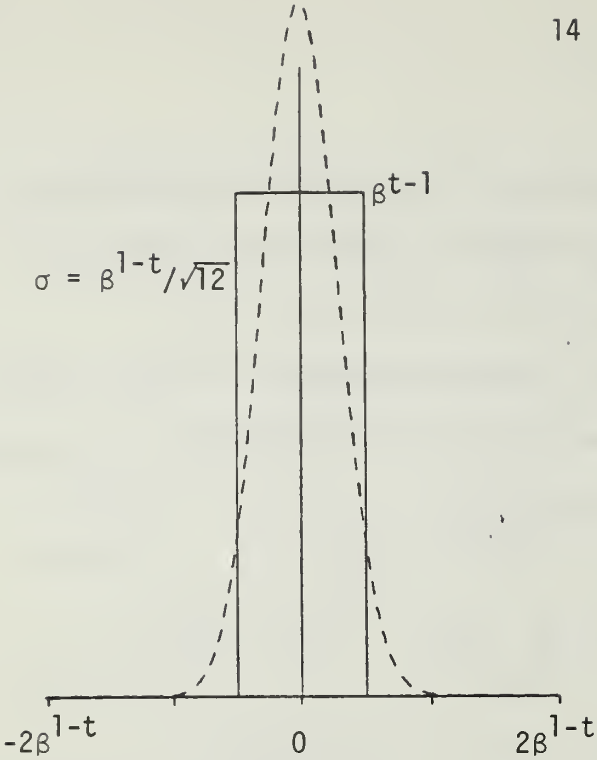


Figure 1.
p_R for Rounding and its normal approximation

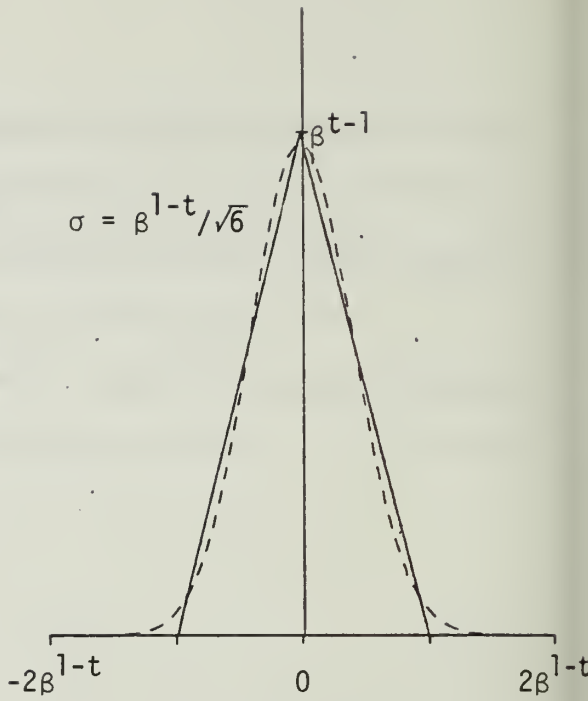


Figure 2.
p_R*p_R and normal approximation

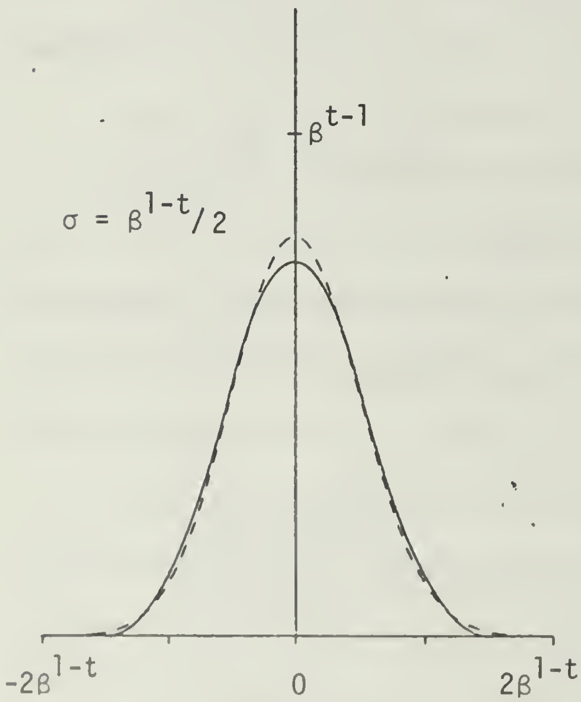


Figure 3.
$p_R*p_R*p_R$ and normal approximation

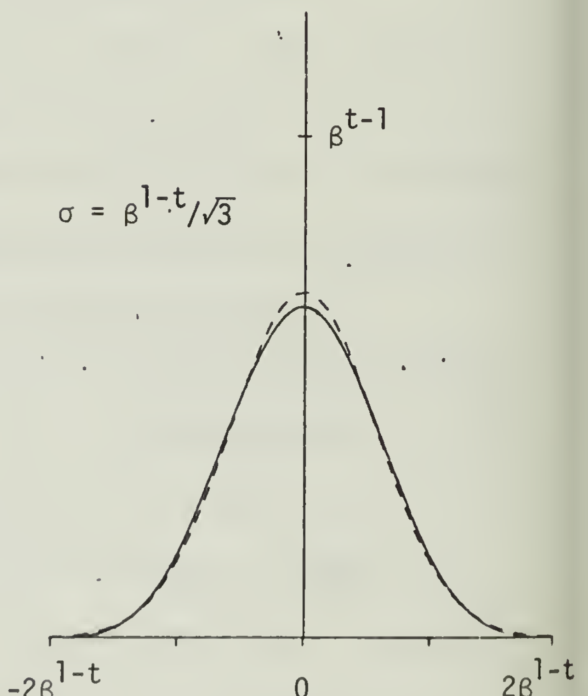


Figure 4.
$p_R*p_R*p_R*p_R$ and normal approximation

Obviously what was said in the previous paragraph holds without modification for Rounding. For chopping we must simply be more careful in computation of the mean μ, since the mean of $\overset{\vee}{p}_R$ is $+ \frac{1}{2}\beta^{1-t}$ instead of $- \frac{1}{2}\beta^{1-t}$ as it was before. For Chopping, division errors tend to cancel out multiplication errors since they are of the opposite sign--this is reflected here by the fact that division makes the mean of the final error more positive, while multiplication makes it more negative.

We commented in Section 2 that addition and subtraction were not as nice as multiplication and division insofar as their results were not easily expressible as smeared numbers. For the rest of this section we address the problem of reducing a sum of smeared numbers to a single smeared number.

It is expedient to work in absolute errors instead of the relative errors we have been working with until now. The following definition and lemma provide the necessary groundwork.

Def: For a bounded density f and nonzero real number A, the stretched density f^A is given by

$$f^A(x) = \frac{1}{|A|} f\left(\frac{x}{A}\right) \tag{3.11}$$

We say that f^A is f stretched by the value A. Here we have restricted our definition to bounded densities to avoid the problem introduced by impulse functions.

Lemma 1 If f is a bounded density, then $[A,f] = [1,f^A]$.

<u>Proof</u> Since $\int_{-\infty}^{A\cdot C} f^A(t)dt = \int_{-\infty}^{A\cdot C} \frac{1}{|A|} f(\frac{t}{A})dt\Big|_{\mu=t/A} = \int_{-\infty}^{C} f(\mu)d\mu$.

$[1,f^A]$ is equivalent to $[A,f]$ under the definition of smeared numbers. Notice that $[1,f^A]$ is an "absolute" smeared number, no longer relative to A. Effectively $[1,f^A] = f^A$.

<u>Theorem 2</u> If $fl(A + B) = [A, 1 * p_1] + [B, 1 * p_2]$ and $A \neq 0$, $B \neq 0$, $A + B \neq 0$ then $fl(A + B) = [A + B, 1 * p]$ where $p(x)$

$$= p_1^{(A/A+B)} * p_2^{(B/A+B)}(x) \tag{3.12}$$

<u>Proof</u> Brute force. Note that $(1 * f)(x) = f(x - 1)$ and $(1 * f)^A(x) = \frac{1}{|A|} f(\frac{x}{A} - 1)$. Now:

$$\begin{aligned} fl(A + B) &= [A, 1 * p_1] + [B, 1 * p_2] \\ &= [1, (1 * p_1)^A] + [1, (1 * p_2)^B] \\ &= (1 * p_1)^A * (1 * p_2)^B \qquad \text{(Lemma 1)} \end{aligned}$$

(This follows from the remark that $[1,f] = f$, and observation 2, Part 2.)

$$\begin{aligned} &= \int_{-\infty}^{\infty} \frac{1}{|A|} p_1(\frac{x-t}{A} - 1) \frac{1}{|B|} p_2(\frac{t}{B} - 1)\, dt \Big|_{t=(A+B)\mu} \\ &= \left|\frac{A + B}{AB}\right| \int_{-\infty}^{\infty} p_1(\frac{x - (A + B)\mu - A}{A}) \\ &\qquad p_2(\frac{(A + B)\, \mu - B}{B})\, d_\mu \Big|_{\mu=v+(B/A+B)} \\ &= \left|\frac{A + B}{AB}\right| \int_{-\infty}^{\infty} p_1(\frac{x - (A + B)v - (A + B)}{A}) \\ &\qquad p_2(\frac{(A + B)v}{B})\, dv \end{aligned}$$

$$= \left|\frac{A + B}{AB}\right| \int_{-\infty}^{\infty} \left|\frac{A}{A + B}\right| p_1^{(A/A+B)}\left(\frac{x}{A + B} - 1 - v\right)$$

$$\left|\frac{B}{A + B}\right| \; p_2^{(B/A+B)}(v) \; dv$$

$$= \frac{1}{|A + B|} p_1^{(A/A+B)} * p_2^{(B/A+B)}\left(\frac{x}{A + B} - 1\right)$$

$$= \frac{1}{|A + B|} \; 1 * p_1^{(A/A+B)} * p_2^{(B/A+B)}\left(\frac{x}{A + B}\right)$$

$$= (1 * p_1^{(A/A+B)} * p_2^{(B/A+B)})^{A+B}$$

$$= [1, (1 * p_1^{(A/A+B)} * p_2^{(B/\ A+B)})^{A+B}]$$

$$= [A + B, 1 * p_1^{(A/A+B)} * p_2^{(B/A+B)}]$$

Notice that $A - B = A + (-B)$ so Theorem 2 applies to subtraction also. Theorem 2 leaves us a few steps short of finishing off all the problems of addition.

<u>Corollary 1</u>.

If $\qquad fl\left(\sum_{i=1}^{n} A_i\right) = [A_1, 1 * p_1] + [A_2, 1 * p_2] + \ldots + [A_n, 1 * p_n]$

then $\qquad fl\left(\sum_{i=1}^{n} A_i\right) = \left[\sum_{i=1}^{n} A_i, 1 * p\right]$

where $\qquad p = p_1^{(A_1/\Sigma A_i)} * p_2^{(A_2/\Sigma A_i)} * \ldots * p_n^{(A_n/\Sigma A_i)} \qquad (3.13)$

(<u>Proof</u>. Trivial. Naturally we now require that $A_i \neq 0$ $(i = 1, \ldots, n)$ and $\sum_{i=1}^{n} A_i \neq 0$.)

Theorem 3. Let the mean and variance for the density f be μ and σ^2, respectively. Then the mean and variance for f^A are $(A\mu)$ and $(A^2\sigma^2)$, respectively. (Proof. Trivial.)

From corallary 1 and Theorem 3 it is immediately obvious how to apply the central limit theorem to addition and subtraction. If the relative error density p_1 of A has mean μ_1 and variance σ_1^2 and the density p_2 of B has mean μ_2 and variance σ_2^2, then the density p of A + B (as in Theorem 1) has

$$\text{mean} \qquad \mu = \left(\frac{A}{A+B}\right)\mu_1 + \left(\frac{B}{A+B}\right)\mu_2 \tag{3.14}$$

$$\text{and variance } \sigma^2 = \left(\frac{A}{A+B}\right)^2\sigma_1^2 + \left(\frac{B}{A+B}\right)^2\sigma_2^2 \tag{3.15}$$

After many additions the error density will converge to a normal probability density--but more operations are needed because the stretched densities $p_j^{(A_j/\Sigma A_i)}$ will not be identical up to translation as they were for multiplication: accumulated errors in multiplication become normally distributed much more quickly than those in addition. The reader who has researched the Central Limit Theorem has probably discovered that most of the existing proofs assume identically distributed summand variables. Unidentical distributions can slow down convergence--especially when the variables differ greatly. Thus when stretching factors like A/(A - B) became large (implying lots of cancellation and loss of significance in the computation of A - B), convergence toward normal error densities is slowed.

In the next section we show that we can guarantee confidence intervals almost as good as $[\mu - 3\sigma, \mu + 3\sigma]$ all of the time, even with

slowed convergence. Thus convergence and the Central Limit Theorem in general should be used only for an intuitive feel of what is going on, while the growth of σ is worth serious attention. Note that when stretching factors like $A/(A - B)$ get large, convergence not only slows down but variances get large also (Theorem 3). This is a serious weakness of statistical error analysis.

4. PROOF THAT STATISTICAL BOUNDS HOLD FOR ROUNDOFF ERRORS

Section 3 introduced the Central Limit Theorem, showed that it could be used to show how accumulated roundoff errors have asymptotically normal distributions, and discussed the rate of convergence to normal distributions. The literature surrounding the Central Limit Theorem is labyrinthine since there are any number of assumptions one can make about the summand variables which simplify or aggravate the proof. However, as we mentioned earlier we are not so much concerned with proving convergence as being able to say, for any roundoff variable δ, something like

$$\Pr(|\delta - \mu| \geq 3\sigma) \leq .0028 \tag{4.1}$$

as we can with normally distributed random variables. In general given a fixed small probability ρ we want to find a corresponding λ such that

$$\Pr(|\delta - \mu| = \lambda\sigma) \leq \rho. \tag{4.2}$$

Since δ will correspond to a lengthy convolution of error densities, determining λ exactly will be unfeasible. Instead we can get bounds on λ. It would be simpler to just assume that δ is normally distributed and look up λ in normal distribution tables. However, this is <u>not</u> always legitimate: see e.g. [5]. Although three or four convolutions can approximate a normal density well (see Figures 1-4) large stretching factors can upset this approximation. The purpose of this section is to bound how <u>much</u> these factors can upset it, by obtaining bounds for λ in (4.2). We start with <u>Chebyshev's Inequality</u>.

Theorem 4. Let δ be a random variable with underlying density p(t), having mean 0 and variance σ^2. Then

$$\Pr(|\delta| \geqq \lambda\sigma) \leqq 1/\lambda^2$$

Proof. $\sigma^2 = E[\delta^2]$

$$= \int_{-\infty}^{\infty} t^2 \, p(t)\, dt$$

$$\geqq \int_{|t| \geqq \lambda\sigma} t^2 \, p(t)\, dt$$

$$\geqq (\lambda\sigma)^2 \int_{|t| \geqq \lambda\sigma} p(t)\, dt$$

$$= (\lambda\sigma)^2 \Pr(|\delta| \geqq \lambda\sigma)$$

$$\Rightarrow \Pr(|\delta| \geqq \lambda\sigma) \leqq \frac{1}{\lambda^2}, \text{ as desired.}$$

This says that if we want

$$\Pr(|\delta| \geqq \lambda\sigma) \leqq .0028$$

then we will be guaranteed this relationship provided

$$\frac{1}{\lambda^2} = .0028 \Rightarrow \lambda \geqq 18.89$$

Clearly this is not very good since $\lambda = 3$ works for normally distributed variables, and after many convolutions we expect δ to be very close to normal. It turns out we can obtain a much tighter limit on λ if we go through the analysis used to obtain the Chernoff bound.

Theorem 5. Let $\delta = \sum_{i=1}^{N} \delta_i$ be a roundoff error, i.e., δ is represented by

the convolution of N (possibly stretched) copies of p_R, the rounding density. Then

$$\Pr(|\delta| \geqq \lambda\sigma) \leqq 2e^{-\lambda^2/2}$$

where σ^2 is the variance of δ.

<u>Proof</u>. Set $d(x) = \begin{cases} 0 & x \leq \lambda\sigma \\ 1 & x \geqq \lambda\sigma \end{cases}$

Then

$$d(x) \leqq e(x) \overset{\text{def}}{=\!=} \exp(L(x - \lambda\sigma)) \text{ for any positive } L.$$

This implies

$$\begin{aligned} \Pr(\delta \geqq \lambda\sigma) &= E[d(\delta)] \\ &\leqq E[e(\delta)] \\ &= E[\exp(L\delta)\exp(-L\lambda\sigma)] \\ &= E[\exp(L\delta)]\, e^{-L\lambda\sigma} \end{aligned}$$

Now if $p(t)$ is the density associated with δ then

$$\begin{aligned} E[\exp(L\delta)] &= \int_{-\infty}^{\infty} \exp(Lt)\, p(t)\, dt \\ &= \sum_{n=0}^{\infty} \frac{L^n}{n!} \int_{-\infty}^{\infty} t^n\, p(t)\, dt \quad \text{[p is of compact support]} \\ &= \sum_{n=0}^{\infty} \frac{M_n L^n}{n!} \end{aligned}$$

where M_n is the nth moment of $p(t)$. Since $p(t)$ is symmetric about 0 (i.e., even) the odd terms vanish and we are left with

$$E[\exp(L\delta)] = \sum_{n=0}^{\infty} \frac{M_{2n} L^{2n}}{(2n)!}$$

We now need the following lemma.

<u>Lemma 2</u>. For roundoff densities p(t)

$$\frac{M_{2n}}{(M_2)^n} \leqq \frac{(2n)!}{n!2^n}$$

<u>Proof</u>. It can be shown that if $p(t) = p_1 * p_2 * \ldots * p_N(t)$ and $M_{n,k}$ is the nth moment of p_k, that the nth moment M_n of p is

$$M_n = \sum_{\substack{0 \\ n_1+\ldots+n_N=n}}^{n} \frac{n!}{n_1!n_2! \ldots n_N!} \prod_{k=1}^{N} M_{n_k,k}$$

This is done as in Theorem 1 by comparing the characteristic functions (essentially Fourier transforms) of both sides of the equation $p = \prod_{k=1}^{N} * p_k$. We then use the fact that the p_k are stretched versions of the roundoff density p_R (2.11), with moments

$$M_{n,k} = \begin{cases} \dfrac{[A_k(\beta^{1-t}/2)]^n}{(2n+1)} & n \text{ even} \\ 0 & n \text{ odd} \end{cases}$$

where $p_k = p_R^{(A_k)}$, i.e., A_k is a nonzero stretching factor. Pulling all of this material together leads to the bound of the lemma.

It is interesting that if p(t) is <u>normal</u> with mean 0,

$$\frac{M_{2n}}{(M_2)^n} = \frac{M_{2n}}{\sigma^{2n}} = \frac{(2n)!}{n!\ 2^n} \qquad (M_2 = \sigma^2)$$

--so we have <u>equality</u> with the stated bound. Therefore:

1. The bound is fairly tight, especially as N gets large.
2. The lemma holds for other densities besides stretched versions of p_R.
3. There is probably an elegant proof of this lemma.

It now follows that

$$E[\exp(L\delta)] \leqq \sum_{n=0}^{\infty} \frac{1}{n!} \left(\frac{L^2 M_2}{2}\right)^n = \exp\left(\frac{L^2 \sigma^2}{2}\right)$$

and therefore

$$P(\delta \geqq \lambda\sigma) \leqq \exp\left(\frac{L^2 \sigma^2}{2} - L\lambda\sigma\right) \quad \text{for all positive L.}$$

The Chernoff Bound is now obtained by choosing L such that the right hand side of this expression is optimal. By differentiating it is easy to see that $L = \lambda/\sigma$ is the correct choice.

It now follows immediately that

$$Pr(\delta \geqq \lambda\sigma) \leqq e^{-\lambda^2/2} \quad \text{and}$$

$$Pr(|\delta| = \lambda\sigma) \leqq 2e^{-\lambda^2/2} \quad \text{as stated.}$$

The Chernoff Bound on the probability is <u>much</u> tighter than the one obtained by Chebyshev's Inequality, and is almost as good as having a normal distribution. For comparison we tabulate the λ's obtained from each method:

$Pr(\lvert\delta\rvert \geq \lambda\sigma)$	λ (δ is normal)	Chernoff Bound for λ (δ is roundoff error)	Chebyshev Bound for λ (δ arbitrary)
.10	1.645	2.448	3.162
.05	1.960	2.716	4.472
.02	2.326	3.035	7.072
.01	2.576	3.255	10.00
.002798	3	3.625	18.90
.002	3.090	3.717	22.36
.001	3.290	3.899	31.62
.0002	3.719	4.292	70.71
.00002	4.265	4.799	223.6
.000002	4.753	5.257	707.2

Table 1. Bounds for λ given information about δ

Table 1 shows us that <u>99.7% confidence intervals on the magnitude of δ</u> are $(-3.625\sigma, 3.625\sigma)$; this result compares quite closely with the $(-3\sigma, + 3\sigma)$ bounds we could use if it were known that δ were normal, and has been used in the program that actually does statistical error analysis (Section 6).

5. CORRELATED ERRORS

Until now we have been assuming that all our relative errors were <u>independent</u> random variables. Now we consider the case in which they are <u>not</u> all independent, i.e., some of the errors are related to others in a common expression. This is the case, for example, in the evaluation of $fl(A\cdot A)$. Assuming that

$$fl(A) = A(1 + \delta_A) = [A, 1 * p_A] \tag{5.1}$$

then our prior analysis would claim

$$\begin{aligned} fl(A\cdot A) &= A\cdot A(1 + \delta_A + \delta_A + \delta_R) \\ &= [A\cdot A, 1 * p_A * p_A * p_R]. \end{aligned} \tag{5.2}$$

This is not correct, however; it is too optimistic. The correct expression is

$$\begin{aligned} fl(A\cdot A) &= A\cdot A(1 + 2\delta_A + \delta_R) \\ &= [A\cdot A, \Delta * p_A^{(2)} * p_R] \end{aligned} \tag{5.3}$$

where $p_A^{(2)}$ is p_A stretched by the factor 2, as in (3.11); i.e.,

$$p_A^{(2)}(t) = \frac{1}{2}\, p_A(t/2). \tag{5.4}$$

We remark that <u>the rounding error δ_R (with density p_R) is always initially an independent random variable</u>. Correlation comes from repeated use of an existing error as in the case with δ_A above; new rounding errors are always uncorrelated with anything.

The generalization of the above analysis to the general case is straightforward. Consider a smeared number $\tilde{A}$

$$\tilde{A} = [A, 1 * p_1^{(c_1)} * p_2^{(c_2)} * \ldots * p_N^{(c_N)}] \tag{5.5}$$

where p_k is a probability density ($k = 1, \ldots, N$) and the C_k are stretching factors. Suppose that two distinct error densities p_i, p_j correspond to the same error variable δ. Without loss of generality (convolution is commutative) we can assume these densities are p_1 and p_2; $\tilde{A}$ can then be rewritten as

$$\tilde{A} = [A, 1 * p_1^{(c_1 + c_2)} * p_3^{(c_3)} * \ldots * p_N^{(c_N)}] \tag{5.6}$$

This corresponds to merging the two references to δ into one. Other references to δ should also be merged.

Note that, because convolution of densities (or addition of random variables) is commutative, correlated errors must be merged throughout computation. For example, in computing

$$X = fl(\cdots(((A + B) + C) + D) + E) + \ldots + Z) + A) \tag{5.7}$$

the error δ_A involved in the last step of the addition will be slightly correlated with the error of the rest of the sum; the resulting smeared number is

$$X = [\chi, 1 * p_A^{(2A/\chi)} * p_B^{(B/\chi)} * \ldots * p_Z^{(Z/\chi)}] \tag{5.8}$$

where $\chi = 2A + B + C + \ldots + Z$.

6. THE SNORT SYSTEM AND PROBLEMS OF IMPLEMENTATION

As this project of statistical error analysis progressed, it was hoped that the final results could be automated to perform a posteriori error bound calculation in the spirit of Interval Analysis. Since for small relative errors $\delta = (fl(x) - x)/x$ we have

$$x \approx fl(x)\ (1 - \delta)$$

it seems natural to evaluate fl(x) in the natural way and use statistical bounds on δ to bound the error in fl(x). While Interval Analysis will give results like

$$x \ \varepsilon\ [x_{low}, x_{high}]$$

we would give results like

$$x \ \varepsilon\ [fl(x)\ (1 + \mu - B),\ fl(x)\ (1 + \mu + B)]$$

where $B = (3.625\sigma)$ is the 99.7%-confidence bound on δ, μ and σ^2 being the computed mean and variance of δ viewed as a random variable.

Automation of the statistical analysis was in fact completed by development of SNORTRAN, a PL/I-like language. It is similar in philosophy to PL/I-FORMAC in that SNORTRAN source is preprocessed and expanced into PL/I, which is then handled by the PL/I compiler. The preprocessor, a SNOBOL routine included at the end of this section, is unsophisticated: current need did not warrant sophistication.

The results of several test programs (also included at the end of this section) reflect negatively on continued use of statistical

relative error analysis for almost all problems. There are at least four reasons for the discouragement:

1. There is no way to represent 0 as a smeared number.
2. In subtraction of two numbers with similar values, cancellation causes enormous increase in error variance (stretching factors get large). One or two such cancellations will result in a value of σ so large that "useless" error bounds result.
3. In only the most well-conditioned, stable computations (e.g., repeated multiplication) are statistical bounds competitive with Interval Analysis.
4. Correlation between errors cannot be easily taken into account (as shown in Section 5) so statistical "bounds" ignoring correlation may use smaller stretching factors than they should, and give incorrectly small results.

The actual implementation uses Interval Analysis methods to generate worst-case values of σ and μ, hence statistical bounds generated by the program should be bounds (ignoring correlation effects). There are four basic SNORT commands, all of them keyword-driven for simplicity:

$$\text{INITIALIZE} \begin{Bmatrix} \text{ROUNDED} \\ \text{CHOPPED} \end{Bmatrix};$$

```
SMEAR  <variable> [,<variable>];
EVAL  <variable>  =  <expression>;
PRINT  <variable> [,<variable>];
```

INITIALIZE causes the preprocessor to copy SNORT system declarations and routines into the translated program output, depending on the type of arithmetic being simulated. If rounded arithmetic is being used, the preprocessor copies in the file SNORT.RND, otherwise copying SNORT.CHP. The former is reproduced at the end of this section; the latter, requiring full Interval Analysis to compute error means, was never written.

SMEAR <variable> results in declarations for <variable> which permit it to be used as a smeared number. "<variable>" may be subscripted if desired (e.g., SMEAR A (10,0:5)) and a list of <variable>s generates a list of appropriate declarations.

EVAL <variable> = <expression> causes an SLR(1) parser to break down <expression> into SNORT system subroutine calls. Currently <expression> should only involve the operators +,-,/,* although this could easily be expanded or modified by changing the parser tables. All variables in <variable> and <expression> should be SMEARed--the preprocessor does not currently check.

PRINT provides formatted output of smeared variables. It lists computed values, 3.625σ bounds and the value bounds they imply, and Interval Analysis value bounds.

Four sample runs are included in the following pages, involving algorithms which highlight progressively more serious problems with statistical relative error analysis. The first shows that even in the best case of serial multiplication, statistical analysis does not give better bounds than interval analysis. The other three show

respectively what happens when addition, subtraction, and computed zeroes are added.

Preliminary example: Computation of sin (1.0)

The first test of the SNORTRAN system attempted to compute $\sin(1) = .84147098\ldots$ via the little-used relationship

$$\frac{\sin\theta}{\theta} = \prod_{n=1}^{\infty} \cos\left(\frac{\theta}{2^n}\right).$$

This concentrates on the roundoff properties of multiplication (ignoring computation of the cosines) which we know statistical analysis handles well. Unfortunately the results are not encouraging. Statistical analysis obtains the bounds

$$\sin(1) \in (.84146956,\ .84147235)$$

after 32 factors, while Interval Analysis gets

$$\sin(1) \in (.8414694,\ .8414710).$$

```
PRELIM: PROCEDURE OPTIONS(MAIN) REORDER;
        INITIALIZE ROUNDED;
        /*------------------------------------------------------*/
        /*  EVALUATE  SIN(1.0) USING THE RELATIONSHIP           */
        /*  SIN(X)/X =  COS(X/2) COS(X/4) COS(X/8) COS(X/16)...*/
        /* -- COSINES TREATED AS HAVING REPRESENTATION ERROR.  */
        /* -- SERIES IS TRUNCATED AFTER 32 COSINE FACTORS.     */
        /* -- CORRECT ANSWER IS  0.84147 09848 07896 50665...  */
        /*------------------------------------------------------*/
        SMEAR SIN_OF_A_BIT, COS_FACTOR;
        EVAL SIN_OF_A_BIT = 1.0E0;
                DO N=1 TO 32;
                CALL SETCONSTANT( COS_FACTOR,
                             COS( 2.000000000000000000000E0**(-N))    );
                EVAL SIN_OF_A_BIT = SIN_OF_A_BIT * COS_FACTOR;
                END;
        PRINT SIN_OF_A_BIT;
        END PRELIM;
```

```
SIN_OF_A_BIT        COMPUTED VALUE:     8.414709568023681E-01
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -2.779955685138702E+01  ,    2.779955685138702E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    8.414695624995408E-01  ,    8.414723511051954E-01 )
     INTERVAL ANALYSIS BOUNDS:             (    8.414694E-01           ,    8.414710E-01          ) .
```

Preliminary Example SNORTRAN Source and Output

Example 0: Computation of exp(8.0)

This program involves the use of addition, multiplication, and division by computing exp(8) with the usual power series, truncated at 40 terms. To ten digits $e^8 = 2980.957987$, and at the end statistical analysis claims

$$e^8 \in (2980.9549\ldots, 2980.961\ldots)$$

while Interval Analysis gets

$$e^8 \in (2980.955 \quad , 2980.958)$$

- a slightly tighter bound.

```
EXO: PROCEDURE OPTIONS(MAIN) REORDER;
        INITIALIZE ROUNDED;
        SMEAR X, XN, FACTN, SN, EXPX;
        EVAL X = 8EO;      /*  EVALUATE EXP(8) THE HARD WAY  */
        EVAL XN = 1.0EO;
        EVAL FACTN = 1.0EO;
        EVAL EXPX = 1.0EO;
                DO N=1 TO 40;
                CALL SETCONSTANT(SN,N);
                EVAL XN = XN*X;
                EVAL FACTN = FACTN * SN;
                EVAL EXPX = EXPX + XN/FACTN;
                PRINT EXPX;
                END;
        END EXO;
```

```
EXPX      COMPUTED VALUE:    2.9809580078125000E+03
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.649799942970276E+01   ,    1.649799942970276E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    2.980955076465397E+03   ,    2.980960939159603E+03 )
     INTERVAL ANALYSIS BOUNDS:             (    2.980955E+03            ,    2.980958E+03          )

EXPX      COMPUTED VALUE:    2.9809580078125000E+03
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.678534758090973E+01   ,    1.678534758090973E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    2.980955025409681E+03   ,    2.980960990215318E+03 )
     INTERVAL ANALYSIS BOUNDS:             (    2.980955E+03            ,    2.980958E+03          )

EXPX      COMPUTED VALUE:    2.9809580078125000E+03
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.706785929203033E+01   ,    1.706785929203033E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    2.980954975213299E+03   ,    2.980961040411701E+03 )
     INTERVAL ANALYSIS BOUNDS:             (    2.980955E+03            ,    2.980958E+03          )

EXPX      COMPUTED VALUE:    2.9809580078125000E+03
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.734576964378357E+01   ,    1.734576964378357E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    2.980954925834482E+03   ,    2.980961089790518E+03 )
     INTERVAL ANALYSIS BOUNDS:             (    2.980955E+03            ,    2.980958E+03          )
```

Example 0: exp(8) SNORTRAN Source and Final Page of Output

Example 1: Computation of $\cos(4\pi)$

This problem involves all four arithmetic operations (using again the usual power series) and is not well-disposed to statistical analysis since there is eventually tremendous cancellation between terms. Note that to the first ten terms the computed sum is roughly -309, and the eleventh term is around +387.6, resulting in a factor of 16 increase in variance. Statistical analysis can no longer give decent results: it provides the bounds

$$\cos(4\pi) \in (-.4183606\ldots, 2.4383515\ldots)$$

whereas Interval Analysis gets

$$\cos(4\pi) \in (.7754054 \quad , 1.227057).$$

Neither of these results are good but the statistical bound is terrible. The relative error bound is so big that the assumption that second order effects are negligible is in question.

```
EX1: PROCEDURE OPTIONS(MAIN) REORDER;
        INITIALIZE ROUNDED;
        SMEAR X, X2, XN, FACTN, SN, COSX;
        EVAL X = 12.566370616E0;   /* = 4*PI; WE ARE GOING TO   */
        EVAL X2 = X*X;             /* COMPUTE COS(4*PI) THE HARD,*/
        EVAL XN = 1.0E0;           /* SLOWLY CONVERGING WAY.    */
        EVAL FACTN = 1.0E0;
        EVAL COSX = 1.0E0;
             DO N=2 TO 40 BY 2;
             CALL SETCONSTANT(SN,N);
             EVAL XN = -XN*X2;
             EVAL FACTN = FACTN * SN * (SN-1.E0);
             EVAL COSX = COSX + XN/FACTN;
             PRINT COSX;
             END;
        END EX1;
```

```
COSX        COMPUTED VALUE:   -7.795684814453125E+01
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -7.925490617752075E+00  ,    7.925490617752075E+00 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -7.795688497103861E+01  ,   -7.795681131802388E+01 )
     INTERVAL ANALYSIS BOUNDS:             (   -7.795685E+01           ,   -7.795682E+01          )

COSX        COMPUTED VALUE:    9.610737304687500E+02
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.100492358207703E+01  ,    1.100492358207703E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    9.610731000576638E+02  ,    9.610743608798360E+02 )
     INTERVAL ANALYSIS BOUNDS:             (    9.610728E+02           ,    9.610737E+02          )

COSX        COMPUTED VALUE:   -4.508164062500000E+03
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.477807629108429E+01  ,    1.477807629108429E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -4.508168033480193E+03  ,   -4.508160091519804E+03 )
     INTERVAL ANALYSIS BOUNDS:             (   -4.508164E+03           ,   -4.508156E+03          )

COSX        COMPUTED VALUE:    1.091447265625000E+04
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -2.054601681232452E+01  ,    2.054601681232452E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    1.091445928997167E+04  ,    1.091448602252833E+04 )
     INTERVAL ANALYSIS BOUNDS:             (    1.091445E+04           ,    1.091448E+04          )

COSX        COMPUTED VALUE:   -1.614603515625000E+04
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -2.946258699893951E+01  ,    2.946258699893951E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -1.614606351041588E+04  ,   -1.614600680208412E+04 )
     INTERVAL ANALYSIS BOUNDS:             (   -1.614606E+04           ,   -1.614597E+04          )

COSX        COMPUTED VALUE:    1.622687890625000E+04
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -4.505303657054901E+01  ,    4.505303657054901E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    1.622683533107228E+04  ,    1.622692248142771E+04 )
     INTERVAL ANALYSIS BOUNDS:             (    1.622678E+04           ,    1.622694E+04          )

COSX        COMPUTED VALUE:   -1.186170703125000E+04
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -7.559598541259765E+01  ,    7.559598541259765E+01 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -1.186176047858189E+04  ,   -1.186165358391811E+04 )
     INTERVAL ANALYSIS BOUNDS:             (   -1.186185E+04           ,   -1.186157E+04          )

COSX        COMPUTED VALUE:    6.619847656250000E+03
     3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.461725826263428E+02  ,    1.461725826263428E+02 ) * BETA**(-T)
     IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    6.619789980397908E+03  ,    6.619905332102090E+03 )
     INTERVAL ANALYSIS BOUNDS:             (    6.619652E+03           ,    6.620012E+03          )
```

```
COSX        COMPUTED VALUE:    -2.917707031250000E+03
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -3.383818244934082E+02  ,   3.383818244934082E+02 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -2.917765878657885E+03  ,  -2.917648183642115E+03 )
    INTERVAL ANALYSIS BOUNDS:             (   -2.917922E+03           ,  -2.917508E+03          )

COSX        COMPUTED VALUE:     1.045739746093750E+03
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -9.479242248535156E+02  ,   9.479242248535156E+02 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    1.045680661079990E+03  ,   1.045798831107510E+03 )
    INTERVAL ANALYSIS BOUNDS:             (    1.045512E+03           ,   1.045950E+03          )

COSX        COMPUTED VALUE:    -3.089838867187500E+02
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -3.212560668945313E+03  ,   3.212560668945313E+03 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -3.090430520464068E+02  ,  -3.089247213910931E+02 )
    INTERVAL ANALYSIS BOUNDS:             (   -3.092163E+02           ,  -3.087636E+02          )

COSX        COMPUTED VALUE:     7.856933593750000E+01
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.268161645507813E+04  ,   1.268161645507813E+04 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    7.850994669297399E+01  ,   7.862872518202600E+01 )
    INTERVAL ANALYSIS BOUNDS:             (    7.833521E+01           ,   7.878589E+01          )

COSX        COMPUTED VALUE:    -1.558444213867188E+01
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -6.501621582031250E+04  ,   6.501621582031250E+04 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -1.564483603558105E+01  ,  -1.552404824176269E+01 )
    INTERVAL ANALYSIS BOUNDS:             (   -1.581692E+01           ,  -1.536748E+01          )

COSX        COMPUTED VALUE:     4.082443237304688E+00
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -2.672873359375000E+05  ,   2.672873359375000E+05 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    4.017403520523594E+00  ,   4.147482954085781E+00 )
    INTERVAL ANALYSIS BOUNDS:             (    3.847870E+00           ,   4.299484E+00          )

COSX        COMPUTED VALUE:     5.127058029174804E-01
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -4.132032375000000E+06  ,   4.132032375000000E+06 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    3.864323510822416E-01  ,   6.389792547527193E-01 )
    INTERVAL ANALYSIS BOUNDS:             (    2.781191E-01           ,   7.297649E-01          )

COSX        COMPUTED VALUE:     1.080962181091309E+00
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -3.562749687500000E+06  ,   3.562749687500000E+06 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    8.514129118393434E-01  ,   1.310511450343274E+00 )
    INTERVAL ANALYSIS BOUNDS:             (    8.463724E-01           ,   1.298023E+00          )

COSX        COMPUTED VALUE:     1.000984191894531E+00
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -6.034146000000000E+06  ,   6.034146000000000E+06 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    6.409670855055082E-01  ,   1.361001298283554E+00 )
    INTERVAL ANALYSIS BOUNDS:             (    7.663939E-01           ,   1.218045E+00          )

COSX        COMPUTED VALUE:     1.011007308959961E+00
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -9.460379375000000E+06  ,   9.460379375000000E+06 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    4.405563630156024E-01  ,   1.581458254904319E+00 )
    INTERVAL ANALYSIS BOUNDS:             (    7.764173E-01           ,   1.228068E+00          )

COSX        COMPUTED VALUE:     1.009881973266602E+00
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.499482700000000E+07  ,   1.499482700000000E+07 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    1.072885108261516E-01  ,   1.912475435707051E+00 )
    INTERVAL ANALYSIS BOUNDS:             (    7.752915E-01           ,   1.226943E+00          )

COSX        COMPUTED VALUE:     1.009995460510254E+00
    3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -2.372667987500000E+07  ,   2.372667987500000E+07 ) * BETA**(-T)
    IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -4.183606485563516E-01  ,   2.438351569576859E+00 )
    INTERVAL ANALYSIS BOUNDS:             (    7.754054E-01           ,   1.227057E+00          )
```

Example 1: $\cos(4\pi)$ SNORTRAN Source and Output

Final example: Lanczos' Algorithm

Lanczos' Algorithm is a method for reducing a symmetric matrix to tridiagonal form. Without re-orthogonalization, the vectors of the orthogonal transformation which it generates deviate gradually from orthonormality. The SNORTRAN coding of the algorithm was written before the problem with computed zeroes in statistical relative error analysis was fully appreciated, otherwise it would never have been done. It is included here as an example of how the SNORTRAN system was intended to be used (the PL/I preprocessor-output is included, since the algorithm uses so many SNORTRAN features), and also as an indicator of how severe a problem the irrepresentability of zero is. The program is initially given $v_1 = (1,0,0,0,\ldots)$ and is asked to generate 63 other mutually orthonormal vectors. Statistical bounds for $\langle v_1, v_2 \rangle$ are huge because the result is so close to zero, and during computation of $\langle v_2, v_3 \rangle$ the program blows up because a zero is actually encountered.

```
LANCZOS:  PROC OPTIONS(MAIN) REORDER;
          DECLARE
                  (I,J,K,II,IK,IX,IL,NLIMIT,N,P,NP) FIXED BIN,
                  T  FLOAT BINARY(53);
          SMEAR  U(64), V(8,64), W(64), ALPHA, BETA, GAMMA;
          SMEAR  VSAVE(8,64), IPVAL(8), TEMP1(64), TEMP2(64);
          DECLARE
                  STRING (8) CHAR(13) INIT('<V(J),V(J)>','<V(J),V(J-1)>',
                  '<V(J),V(J-2)>','<V(J),V(J-3)>','<V(J),V(J-4)>',
                  '<V(J),V(J-5)>','<V(J),V(J-6)>','<V(J),V(J-7)>');
          INITIALIZE ROUNDED;
          N = 8; P=8;
          NP = N*P;

  /*                      INIT V(1)                          */
          CALL SETCONSTANT(V(1,1),SQRT(1.000000000000000000000E08/NP));
                  DO K=1 TO NP;
                  VSAVE(1,K) , V(1,K) = V(1,1);
                  END;

  /*                      INIT U(1) = A V(1)                 */
          DO II=1 TO NP;   TEMP1(II) = V(1,II);   END;
          CALL AMULT(U,TEMP1);

  /*                      MAIN LOOP                          */
                  DO IX=2 TO NP;
                  IK = MOD(IX-2,8) + 1;
                  DO II=1 TO NP;   TEMP1(II) = V(IK,II);   END;
                  CALL INNER(ALPHA,TEMP1,U);
                          DO K=1 TO NP;
                          EVAL W(K) = U(K) - ALPHA * V(IK,K);
                          END;
                  CALL INNER(GAMMA,W,W);
```

Computation of the square root of the smeared number GAMMA

```
                  GAMMA.COMPUTED_VALUE=SQRT(GAMMA.COMPUTED_VALUE);
                          GAMMA.VARIANCE = GAMMA.VARIANCE/2;
                          T=GAMMA.IA_LB; GAMMA.IA_LB=ROUNDDOWN(SQRT(T));
                          T=GAMMA.IA_UB; GAMMA.IA_UB=ROUNDUP  (SQRT(T));

                  IL = MOD(IK,8) + 1;
                          DO K=1 TO NP;
                          EVAL V(IL,K) = W(K) / GAMMA;
                          END;

                  BETA = GAMMA;

                  DO II=1 TO NP;    TEMP1(II) = V(IL,II);   END;
                  CALL AMULT(U,TEMP1);
                          DO K=1 TO NP;
                          EVAL U(K) = U(K) - BETA * V(IK,K);
                          END;

  /*                      END OF MAIN LOOP COMPUTATION                 */
  /*                      NOW CHECK INNER PRODUCTS...                  */

                  IF IX<8 THEN NLIMIT=IX; ELSE NLIMIT=8;
                  PUT SKIP EDIT ('J = ',IX) (SKIP(2),A,F(3));
                          DO I=1 TO NLIMIT;
                          J = MOD(IL+8-I,8) + 1;
                          DO II=1 TO NP; TEMP1(II) = V(IL,II);  END;
                          DO II=1 TO NP; TEMP2(II) = V(J,II);   END;
```

```
                    CALL INNER(IPVAL(I),TEMP1,TEMP2);
                    CALL SMEARED_PRINT(STRING(I),IPVAL(I));
                    END;
            END;

AMULT:      PROC(U,V);
            SMEAR U(*), V(*);
            DECLARE (I,J,K) FIXED BIN;
            K = 0;
                    DO J=1 TO P;
                    DO I=1 TO N;
                    K = K + 1;
                    EVAL  U(K) = 4 * V(K);
                    IF I>1 THEN  EVAL   U(K) = U(K) - V(K-1);
                    IF I<N THEN  EVAL   U(K) = U(K) - V(K+1);
                    IF J>1 THEN  EVAL   U(K) = U(K) - V(K-N);
                    IF J<P THEN  EVAL   U(K) = U(K) - V(K+N);
                    END;
                    END;
            END AMULT;


INNER:      PROC(IP,U,V);
            SMEAR IP, U(*), V(*);
            DECLARE K FIXED BIN;
            IP = 0;
                    DO K=1 TO NP;
                    EVAL IP = IP + U(K) * V(K);
                    END;
            END INNER;

END LANCZOS;
```

Final Example: Lanczos' Algorithm SNORTRAN Source

Preprocessor Output of LANCZOS Program

```
MT LEV NT

 1      0  |LANCZOS:  PROC OPTIONS(MAIN) REORDER;
 2   1  0  |      DECLARE
           |               (I,J,K,II,IK,IX,IL,NLIMIT,N,P,NP) FIXED BIN,
           |               T  FLOAT BINARY(53);
           |
 3   1  0  | DECLARE
           |    1 U(64) LIKE SMEARED_NUMBER,
           |    1 V(8,64) LIKE SMEARED_NUMBER,
           |    1 W(64) LIKE SMEARED_NUMBER,
           |    1 ALPHA LIKE SMEARED_NUMBER,
           |    1 BETA LIKE SMEARED_NUMBER,
           |    1 GAMMA LIKE SMEARED_NUMBER;
           |
 4   1  0  | DECLARE
           |    1 VSAVE(8,64) LIKE SMEARED_NUMBER,
           |    1 IPVAL(8) LIKE SMEARED_NUMBER,
           |    1 TEMP1(64) LIKE SMEARED_NUMBER,
           |    1 TEMP2(64) LIKE SMEARED_NUMBER;
 5   1  0  |      DECLARE
           |               STRING (8) CHAR(13) INIT('<V(J),V(J)>','<V(J),V(J-1)>',
           |               '<V(J),V(J-2)>','<V(J),V(J-3)>','<V(J),V(J-4)>',
           |               '<V(J),V(J-5)>','<V(J),V(J-6)>','<V(J),V(J-7)>');
           |
           |
 6   1  0  |      DECLARE
           |         1     SMEARED_NUMBER,
           |           2  COMPUTED_VALUE  FLOAT BINARY(21),
           |           2  VARIANCE        FLOAT BINARY(21), /*UNITS BETA**(-2T)*/
           |           2  INTDATA,
           |              3   IA_LB        FLOAT BINARY(21), /* INTERVAL         */
           |              3   IA_UB        FLOAT BINARY(21); /*  ANALYSIS DATA   */
           |
 7   1  0  |      DECLARE
           |               1 REG(10) LIKE SMEARED_NUMBER,
           |               BETA_MT  FLOAT BINARY(21) INIT(1.0E-24B), /*16**(-6)*/
           |               HUGE     FLOAT BINARY(21) INIT(1.0E+88B);
           |
 8   1  0  |SMEARED_ADD:  PROC (A,B,C);
           |      /*  PERFORMS ADDITION  A = B + C  FOR SMEARED NUMBERS  */
 9   2  0  |      DECLARE 1 (A,B,C) LIKE SMEARED_NUMBER,
           |              (T,T1,T2,T3) FLOAT BINARY(53);
10   2  0  |      T = B.COMPUTED_VALUE;
11   2  0  |      A.COMPUTED_VALUE = ROUND(T + C.COMPUTED_VALUE);
12   2  0  |      T2 = MAXSQ(B.INTDATA);
13   2  0  |      T3 = MAXSQ(C.INTDATA);
14   2  0  |      CALL INTERVAL(A.INTDATA,1 /*ADD*/, B.INTDATA,C.INTDATA);
15   2  0  |      T1 = MINSQ(A.INTDATA);
16   2  0  |      IF ((T2+T3)/T1) > HUGE  THEN
17   2  1  |              DO;  A.VARIANCE = HUGE; RETURN;  END;
20   2  0  |      A.VARIANCE = ROUNDUP( T2/T1*B.VARIANCE + T3/T1*C.VARIANCE );
21   2  0  |      CALL ADD_ROUNDING_ERROR(A);
22   2  0  |      END SMEARED_ADD;
           |
```

```
23    1  0  |SMEARED_NEGATE: PROC (A,B);
            |        /*  PERFORMS NEGATION  A = -B  FOR SMEARED NUMBERS  */
24    2  0  |        DECLARE  1 (A,B) LIKE SMEARED_NUMBER,
            |                 T FLOAT BINARY(21);
25    2  0  |        A.COMPUTED_VALUE = -B.COMPUTED_VALUE;
26    2  0  |        A.VARIANCE = B.VARIANCE;
27    2  0  |        T = B.IA_LB;
28    2  0  |        A.IA_LB = -B.IA_UB;
29    2  0  |        A.IA_UB = -T;
30    2  0  |        END SMEARED_NEGATE;
            |
31    1  0  |SMEARED_SUB:  PROC (A,B,C);
            |        /*  PERFORMS SUBTRACTION  A = B - C  FOR SMEARED NUMBERS  */
32    2  0  |        DECLARE 1 (A,B,C) LIKE SMEARED_NUMBER,
            |                (T,T1,T2,T3) FLOAT BINARY(53);
33    2  0  |        T = B.COMPUTED_VALUE;
34    2  0  |        A.COMPUTED_VALUE = ROUND(T - C.COMPUTED_VALUE);
35    2  0  |        T2 = MAXSQ(B.INTDATA);
36    2  0  |        T3 = MAXSQ(C.INTDATA);
37    2  0  |        CALL INTERVAL(A.INTDATA,2 /*SUB*/, B.INTDATA,C.INTDATA);
38    2  0  |        T1 = MINSQ(A.INTDATA);
39    2  0  |        IF ((T2+T3)/T1) > HUGE  THEN
40    2  1  |                DO;  A.VARIANCE = HUGE; RETURN;  END;
43    2  0  |        A.VARIANCE = ROUNDUP( T2/T1*B.VARIANCE + T3/T1*C.VARIANCE );
44    2  0  |        CALL ADD_ROUNDING_ERROR(A);
45    2  0  |        END SMEARED_SUB;
            |
46    1  0  |SMEARED_MULT:  PROC(A,B,C);
            |        /*  PERFORMS MULTIPLICATION  A = B * C  FOR SMEARED NUMBERS  */
47    2  0  |        DECLARE  1  (A,B,C) LIKE SMEARED_NUMBER,
            |                 T FLOAT BINARY(53);
48    2  0  |        T = B.COMPUTED_VALUE;
49    2  0  |        A.COMPUTED_VALUE = ROUND(T * C.COMPUTED_VALUE);
50    2  0  |        CALL INTERVAL(A.INTDATA,3 /*MULT*/,B.INTDATA,C.INTDATA);
51    2  0  |        T = B.VARIANCE;
52    2  0  |        A.VARIANCE = ROUNDUP(T + C.VARIANCE);
53    2  0  |        CALL ADD_ROUNDING_ERROR(A);
54    2  0  |        END SMEARED_MULT;
            |
55    1  0  |SMEARED_DIV:  PROC(A,B,C);
            |        /*  PERFORMS DIVISION  A = B / C  FOR SMEARED NUMBERS  */
56    2  0  |        DECLARE  1  (A,B,C) LIKE SMEARED_NUMBER,
            |                 T FLOAT BINARY(53);
57    2  0  |        T = B.COMPUTED_VALUE;
58    2  0  |        A.COMPUTED_VALUE = ROUND(T / C.COMPUTED_VALUE);
59    2  0  |        CALL INTERVAL(A.INTDATA,4 /*DIV*/,B.INTDATA,C.INTDATA);
60    2  0  |        T = B.VARIANCE;
61    2  0  |        A.VARIANCE = ROUNDUP(T + C.VARIANCE);
62    2  0  |        CALL ADD_ROUNDING_ERROR(A);
63    2  0  |        END SMEARED_DIV;
            |
64    1  0  |SETCONSTANT: PROC(A,C);
            |        /*  SETS SMEARED NUMBER  A  TO THE INITIAL VALUE  C  */
65    2  0  |        DECLARE  1  A  LIKE SMEARED_NUMBER,
            |                 C FLOAT BINARY(53);
66    2  0  |        A.COMPUTED_VALUE = ROUND(C);
67    2  0  |        A.IA_LB = ROUNDDOWN(C);
68    2  0  |        A.IA_UB = ROUNDUP(C);
69    2  0  |        A.VARIANCE = 0E0B;
70    2  0  |        IF C ¬= PRECISION(C,21) THEN
            |                CALL ADD_ROUNDING_ERROR(A);
71    2  0  |        END SETCONSTANT;
            |
```

```
72   1  0  |INTERVAL:  PROC(A,OP,B,C);
           |        /*  PERFORMS INTERVAL ARITHMETIC   A = B OP C   */
73   2  0  |        DECLARE 1 INTERVAL,
           |                  2 IA_LB      FLOAT BINARY(21),
           |                  2 IA_UB      FLOAT BINARY(21),
           |             1 (A,B,C) LIKE INTERVAL,
           |             (S1,S2,S3,S4) FLOAT BINARY(53),
           |             IOP(4) LABEL,
           |             OP      FIXED BINARY(15);
74   2  0  |        S1 = B.IA_LB; S2 = B.IA_UB; S3 = C.IA_LB; S4 = C.IA_UB;
78   2  0  |        GO TO IOP(OP);
           |
79   2  0  |        IOP(1):       /*  ADDITION  */
           |        A.IA_LB = ROUNDDOWN(S1 + S3);
80   2  0  |        A.IA_UB = ROUNDUP  (S2 + S4);
81   2  0  |        RETURN;
           |
82   2  0  |        IOP(2):     /*  SUBTRACTION  */
           |        A.IA_LB = ROUNDDOWN(S1 - S4);
83   2  0  |        A.IA_UB = ROUNDUP  (S2 - S3);
84   2  0  |        RETURN;
           |
85   2  0  |        IOP(3):     /*  MULTIPLICATION  */
           |        A.IA_LB = MIN(    ROUNDDOWN(S1 * S3),
           |                          ROUNDDOWN(S1 * S4),
           |                          ROUNDDOWN(S2 * S3),
           |                          ROUNDDOWN(S2 * S4)    );
86   2  0  |        A.IA_UB = MAX(    ROUNDUP  (S1 * S3),
           |                          ROUNDUP  (S1 * S4),
           |                          ROUNDUP  (S2 * S3),
           |                          ROUNDUP  (S2 * S4)    );
87   2  0  |        RETURN;
           |
88   2  0  |        IOP(4):     /*  DIVISION  */
           |        A.IA_LB = MIN(    ROUNDDOWN(S1 / S3),
           |                          ROUNDDOWN(S1 / S4),
           |                          ROUNDDOWN(S2 / S3),
           |                          ROUNDDOWN(S2 / S4)    );
89   2  0  |        A.IA_UB = MAX(    ROUNDUP  (S1 / S3),
           |                          ROUNDUP  (S1 / S4),
           |                          ROUNDUP  (S2 / S3),
           |                          ROUNDUP  (S2 / S4)    );
90   2  0  |        RETURN;
           |
91   2  0  |        END INTERVAL;
           |
92   1  0  |MAXSQ:  PROC(A)  RETURNS(FLOAT BINARY(53));
           |        /*  RETURNS MAX VALUE OF A**2 OVER INTERVAL A  */
93   2  0  |        DECLARE 1 A,  /* INTERVAL  */
           |                 2 LB FLOAT BINARY(21),
           |                 2 UB FLOAT BINARY(21),
           |                 (T1,T2) FLOAT BINARY(53);
94   2  0  |        T1 = PRECISION(A.LB,53)**2;
95   2  0  |        T2 = PRECISION(A.UB,53)**2;
96   2  0  |        T1 = MAX(T1,T2);
97   2  0  |        RETURN(T1);
98   2  0  |        END MAXSQ;
           |
```

```
 99   1  0  |MINSQ:  PROC(A)  RETURNS(FLOAT BINARY(53));
            |       /*  RETURNS MIN VALUE OF A**2 OVER INTERVAL A  */
100   2  0  |       DECLARE 1 A,  /* INTERVAL  */
            |               2 LB FLOAT BINARY(21),
            |               2 UB FLOAT BINARY(21),
            |                 (T1,T2) FLOAT BINARY(53);
101   2  0  |       IF (A.LB*A.UB <= 0E0B) THEN   /*  INTERVAL CONTAINS ZERO  */
102   2  1  |               DO;  T1 = 1E0B/HUGE;  RETURN(T1); END;
105   2  0  |       T1 = PRECISION(A.LB,53)**2;
106   2  0  |       T2 = PRECISION(A.UB,53)**2;
107   2  0  |       T1 = MIN(T1,T2);
108   2  0  |       RETURN(T1);
109   2  0  |       END MINSQ;
            |
            |
110   1  0  |ADD_ROUNDING_ERROR: PROC(A);
111   2  0  |       DECLARE 1 A LIKE SMEARED_NUMBER,
            |               (D,S) FLOAT BINARY(53);
            |       /*  COMPUTE       D = MIN(MANTISSA(A))                    */
            |       /*                S =  1 / (2D)**2  / 3                   */
            |       /*                   = COMPUTATION ROUNDOFF ERROR VARIANCE, */
            |       /*                     UNITS BETA**(-2T).                 */
            |
112   2  0  |       IF A.VARIANCE >= HUGE | A.COMPUTED_VALUE=0E0B THEN RETURN;
113   2  0  |       D = MIN(MANTISSA(A.IA_LB),MANTISSA(A.IA_UB));
114   2  0  |       S = 1E0B / (D*D*12E0);
115   2  0  |       A.VARIANCE = ROUNDUP(A.VARIANCE + D);
116   2  0  |       END ADD_ROUNDING_ERROR;
            |
            |
117   1  0  | MANTISSA:  PROC(X)  RETURNS(FLOAT BINARY(53));
118   2  0  |       DECLARE (X,Y) FLOAT BINARY(21);
119   2  0  |       Y = ABS(X);
120   2  0  |               DO WHILE (Y > 1.0000000000000000000000E0B);
121   2  1  |               Y = Y * 1.0000000000000000000000E-4B;
122   2  1  |               END;
123   2  0  |               DO WHILE (Y < 1.0000000000000000000000E-4B);
124   2  1  |               Y = Y * 1.0000000000000000000000E+4B;
125   2  1  |               END;
126   2  0  |       RETURN(Y);
127   2  0  |       END MANTISSA;
            |
128   1  0  | SMEARED_PRINT:  PROC(AA,A);
129   2  0  |       DECLARE 1 A LIKE SMEARED_NUMBER,
            |               AA  CHAR(*),
            |               (S,S1,S2,ABS1,ABS2,ABS3,ABS4) FLOAT BIN(53);
130   2  0  |       S = SQRT(A.VARIANCE);
131   2  0  |       S1 = -3.625 * S;   /* 99.7% CONFIDENCE LIMITS  */
132   2  0  |       S2 = +3.625 * S;
133   2  0  |       ABS3 = A.COMPUTED_VALUE * (1 + S1*BETA_MT);
134   2  0  |       ABS4 = A.COMPUTED_VALUE * (1 + S2*BETA_MT);
135   2  0  |       IF A.COMPUTED_VALUE < 0  THEN
136   2  1  |               DO;  ABS1=ABS3; ABS3=ABS4; ABS4=ABS1;  END;
            |
140   2  0  |       PUT FILE(SYSPRINT) SKIP EDIT
            |
            |       (AA,'COMPUTED VALUE:',A.COMPUTED_VALUE)
            |               (SKIP,COL(7),A,X(8),A,E(25,15,16))
            |
            |       ('3.625*SIGMA BOUNDS ON RELATIVE ERROR:','(',S1,'  ,',S2,
            |         ' ) * BETA**(-T)' )
            |               (SKIP,COL(12),A,COL(50),A,2 (E(25,15,16),A))
            |
```

```
                ('IMPLIED 99.7%-CONFIDENCE BOUNDS:', '(', ABS3, '  ,',
                        ABS4, ' )'  )
                        (SKIP,COL(12),A,COL(50),A,2 (E(25,15,16),A))

                ('INTEPVAL ANALYSIS BOUNDS:', '(',A.IA_LB,'  ,', A.IA_UB,' )')
                        (SKIP,COL(12),A,COL(50),A,2 (E(16,6,7),X(9),A));

41  2  0        END SMEARED_PRINT;


42  1  0  ROUND: PROC(X) RETURNS(FLOAT BINARY(21));
43  2  0        DECLARE (X,HALF) FLOAT BINARY(53),
                        RX       FLOAT BINARY(21),
                        E        BIT(64),
                        EEXPT    BIT(8) DEFINED E POSITION(1),
                        RBIT     BIT(1) DEFINED E POSITION(33),
                        F        BIT(64) INIT(
           '0000000000000000000000000000000010000000000000000000000000000000'B),
                        FEXPT    BIT(8) DEFINED F POSITION(1);
44  2  0        E = UNSPEC(X);
45  2  0        IF  RBIT  THEN  DO;       /*   ROUND   */
46  2  1                FEXPT = EEXPT;
47  2  1                UNSPEC(HALF) = F;
48  2  1                RX = X + HALF;
49  2  1                END;
50  2  0            ELSE   RX = X;          /*  TRUNCATE  */
51  2  0        RETURN(RX);
52  2  0        END ROUND;

53  1  0  ROUNDUP: PROC(X) RETURNS(FLOAT BINARY(21));
54  2  0        DECLARE X FLOAT BINARY(53),
                        RX FLOAT BINARY(21);
55  2  0        IF X > 0E0B  THEN RETURN(ROUND(X));
56  2  0                     ELSE DO; RX=X; RETURN(RX); END;
60  2  0        END ROUNDUP;

61  1  0  ROUNDDOWN: PROC(X) RETURNS(FLOAT BINARY(21));
62  2  0        DECLARE X FLOAT BINARY(53),
                        RX FLOAT BINARY(21);
63  2  0        IF X < 0E0B  THEN RETURN(ROUND(X));
64  2  0                     ELSE DO; RX=X; RETURN(RX); END;
68  2  0        END ROUNDDOWN;
69  1  0        N = 8; P=8;
71  1  0        NP = N*P;
          /*                    INIT V(1)                           */
72  1  0        CALL SETCONSTANT(V(1,1),SQRT(1.000000000000000000000E0B/NP));
73  1  0                DO K=1 TO NP;
74  1  1                VSAVE(1,K) , V(1,K) = V(1,1);
75  1  1                END;
          /*                    INIT U(1) = A V(1)                  */
76  1  0        DO II=1 TO NP;   TEMP1(II) = V(1,II);   END;
79  1  0        CALL AMULT(U,TEMP1);
          /*                    MAIN LOOP                           */
80  1  0                DO IX=2 TO NP;
81  1  1                IK = MOD(IX-2,8) + 1;
82  1  1                DO II=1 TO NP;   TEMP1(II) = V(IK,II);   END;
85  1  1                CALL INNER(ALPHA,TEMP1,U);
86  1  1                        DO K=1 TO NP;
```

```
187   1  2  | CALL SMEARED_MULT(REG(1),ALPHA,V(IK,K));
188   1  2  | CALL SMEARED_SUB(W(K),U(K),REG(1));
189   1  2  |                          END;
190   1  1  |                   CALL INNER(GAMMA,W,W);
191   1  1  |                   GAMMA.COMPUTED_VALUE=SQRT(GAMMA.COMPUTED_VALUE);
192   1  1  |                          GAMMA.VARIANCE = GAMMA.VARIANCE/2;
193   1  1  |                          T=GAMMA.IA_LB; GAMMA.IA_LB=ROUNDDOWN(SQRT(T));
195   1  1  |                          T=GAMMA.IA_UB; GAMMA.IA_UB=ROUNDUP  (SQRT(T));
197   1  1  |                   IL = MOD(IK,8) + 1;
198   1  1  |                          DO K=1 TO NP;
            |
199   1  2  | CALL SMEARED_DIV(V(IL,K),W(K),GAMMA);
200   1  2  |                          END;
            |
201   1  1  |                   BETA = GAMMA;
            |
202   1  1  |                   DO II=1 TO NP;    TEMP1(II) = V(IL,II);   END;
205   1  1  |                   CALL AMULT(U,TEMP1);
206   1  1  |                          DO K=1 TO NP;
            |
207   1  2  | CALL SMEARED_MULT(REG(1),BETA,V(IK,K));
208   1  2  | CALL SMEARED_SUB(U(K),U(K),REG(1));
209   1  2  |                          END;
            |/*                        END OF MAIN LOOP COMPUTATION              */
            |/*                        NOW CHECK INNER PRODUCTS...               */
210   1  1  |                   IF IX<8 THEN NLIMIT=IX; ELSE NLIMIT=8;
212   1  1  |                   PUT SKIP EDIT ('J = ',IX) (SKIP(2),A,F(3));
213   1  1  |                          DO I=1 TO NLIMIT;
214   1  2  |                          J = MOD(IL+8-I,8) + 1;
215   1  2  |                          DO II=1 TO NP; TEMP1(II) = V(IL,II);  END;
218   1  2  |                          DO II=1 TO NP; TEMP2(II) = V(J,II);   END;
221   1  2  |                          CALL INNER(IPVAL(I),TEMP1,TEMP2);
222   1  2  |                          CALL SMEARED_PRINT(STRING(I),IPVAL(I));
223   1  2  |                          END;
224   1  1  |                   END;
225   1  0  |            AMULT:  PROC(U,V);
            |
226   2  0  | DECLARE
            |     1 U(*) LIKE SMEARED_NUMBER,
            |     1 V(*) LIKE SMEARED_NUMBER;
227   2  0  |                   DECLARE (I,J,K) FIXED BIN;
228   2  0  |                   K = 0;
229   2  0  |                          DO J=1 TO P;
230   2  1  |                          DO I=1 TO N;
231   2  2  |                          K = K + 1;
            |
232   2  2  | CALL SETCONSTANT(REG(1),4);
233   2  2  | CALL SMEARED_MULT(U(K),REG(1),V(K));
234   2  2  |                          IF I>1 THEN
            | CALL SMEARED_SUB(U(K),U(K),V(K-1));
235   2  2  |                          IF I<N THEN
            | CALL SMEARED_SUB(U(K),U(K),V(K+1));
236   2  2  |                          IF J>1 THEN
            | CALL SMEARED_SUB(U(K),U(K),V(K-N));
237   2  2  |                          IF J<P THEN
            | CALL SMEARED_SUB(U(K),U(K),V(K+N));
238   2  2  |                          END;
239   2  1  |                          END;
240   2  0  |                   END AMULT;
```

```
241   1  0  |      INNER:  PROC(IP,U,V);                                  |
            |                                                             |
242   2  0  | DECLARE                                                     |
            |    1 IP LIKE SMEARED_NUMBER,                                |
            |    1 U(*) LIKE SMEARED_NUMBER,                              |
            |    1 V(*) LIKE SMEARED_NUMBER;                              |
243   2  0  |            DECLARE K FIXED BIN;                             |
244   2  0  |            IP = 0;                                          |
245   2  0  |                    DO K=1 TO NP;                            |
            |                                                             |
246   2  1  | CALL SMEARED_MULT(REG(1),U(K),V(K));                        |
247   2  1  | CALL SMEARED_ADD(IP,IP,REG(1));                             |
248   2  1  |                    END;                                     |
249   2  0  |            END INNER;                                       |
            |                                                             |
250   1  0  |      END LANCZOS;                                           |


J =  2

   <V(J),V(J)>          COMPUTED VALUE:    1.000000953674316E+00
      3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -1.335337650775909E+01  ,   1.335337650775909E+01 ) * BETA**(-T)
      IMPLIED 99.7%-CONFIDENCE BOUNDS:      (    1.000000157750294E+00  ,   1.000001749598339E+00 )
      INTERVAL ANALYSIS BOUNDS:             (    9.999974E-01           ,   1.000001E+00          )

   <V(J),V(J-1)>        COMPUTED VALUE:   -1.043081283569336E-07
      3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -2.450712816696117E+29  ,   2.450712816696117E+29 ) * BETA**(-T)
      IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -1.523669165670400E+15  ,   1.523669165670400E+15 )
      INTERVAL ANALYSIS BOUNDS:             (   -7.003546E-07           ,   6.929040E-07          )


J =  3

   <V(J),V(J)>          COMPUTED VALUE:    9.999997615814208E-01
      3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -2.507748900000000E+07  ,   2.507748900000000E+07 ) * BETA**(-T)
      IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -4.947347057497921E-01  ,   2.494734228912634E+00 )
      INTERVAL ANALYSIS BOUNDS:             (    9.999435E-01           ,   1.000060E+00          )

   <V(J),V(J-1)>        COMPUTED VALUE:    1.132488250732422E-06
      3.625*SIGMA BOUNDS ON RELATIVE ERROR: (   -6.032795910199533E+28  ,   6.032795910199533E+28 ) * BETA**(-T)
      IMPLIED 99.7%-CONFIDENCE BOUNDS:      (   -4.072231344799744E+15  ,   4.072231344799744E+15 )
      INTERVAL ANALYSIS BOUNDS:             (   -1.490116E-05           ,   4.386902E-05          )

IBM461I  'ONCODE'=0300  'OVERFLOW' CONDITION RAISED
   IN STATEMENT 20 AT OFFSET +000288 IN PROCEDURE WITH ENTRY
   SMEARED_ADD
```

Final Example: Preprocessed LANCZOS Program and Output

SNORTRAN System Listings

```
*-----------------------------------------------------------------------*
*                                                                       *
*                     S  N  O  R  T  R  A  N                            *
*                                                                       *
*      A POSTERIORI ROUNDOFF ERROR LANGUAGE PREPROCESSOR                *
*                                                                       *
*-----------------------------------------------------------------------*
        &TRACE = 1000
        DEFINE('FILL(VECTOR,STRING)I,VAL')
        DEFINE('PARSE.EXPR()I,J,LOOKAHEAD.TOKEN,LOOKAHEAD.SYMBOL')
        DEFINE('SEMANTICS(PRODNUM)')
        DEFINE('GETCHAR()')
        DEFINE('GETNONBLANK()')
        DEFINE('SCAN()')
        DEFINE('ADD()')
        DEFINE('REMOVE()')
        DEFINE('EMIT(STRING)')
        DEFINE('LASTEMIT(TARGET)X,Y')
        DEFINE('ALLOCATED.REG()I')
        DEFINE('FREE.REGISTER()I')
        DEFINE('CLEARREGS()I')
        KEYWORD = ('INITIALIZE' | 'SMEAR' | 'EVAL' | 'PRINT') . KEY ' '
        NREGS = 10
        REGISTER = ARRAY(NREGS)
        SEMICOLON = 1
        VARIABLE = 2
        EQUALS = 3
        PLUS = 4
        MINUS = 5
        STAR = 6
        SLASH = 7
        LPAREN = 8
        RPAREN = 9
        FACTOR = 10
        TERM = 11
        EXPR = 12
        ASSMT = 13
        ACCEPT = 14
        COMMA = 20
-EJECT
*
*       SLR(1) EXPRESSION PARSER TABLES
*
        DEPTH = 50
        STATESTACK = ARRAY(DEPTH)
        SYMBOLSTACK = ARRAY(DEPTH)
        TOKENSTACK = ARRAY(DEPTH)
        STORED.DEPTH = 5
        STORED.TOKEN = ARRAY(STORED.DEPTH)
        STORED.SYMBOL = ARRAY(STORED.DEPTH)
        NPRODS = 12;  NTRANS = 76;  NSTATES = 23
        TS = '2,13,3,1,2,4,5,8,10,11,12,2,8,10,11,2,8,10,11,2,'
+            '4,5,8,10,11,12,1,4,5,6,7,9,1,4,5,1,4,5,6,7,'
+            '9,1,4,5,6,7,9,4,5,9,2,8,10,2,8,10,2,8,10,11,'
+            '2,8,10,11,1,4,5,6,7,9,1,4,5,6,7,9,'
        TA = '2,3,4,0,5,6,7,8,9,10,11,5,8,9,12,5,8,9,13,5,'
+            '6,7,8,9,10,14,-5,-5,-5,15,16,-5,-2,17,18,-6,-6,-6,15,16,'
+            '-6,-7,-7,-7,15,16,-7,17,18,19,5,8,20,5,8,21,5,8,9,22,'
+            '5,8,9,23,-3,-3,-3,15,16,-3,-4,-4,-4,15,16,-4,'
        FT = '1,3,4,5,0,12,16,20,0,27,33,36,42,48,51,54,57,61,0,0,'
```

```
+              '0,65,71,'
         LT = '2,3,4,11,-12,15,19,26,-10,32,35,41,47,50,53,56,60,64,-11'
+              ',-8,-9,70,76,'
         LS = '14,13,12,12,12,12,12,11,11,11,10,10,'
         RL = '2,3,3,3,1,2,2,3,3,1,3,1,'
         TRANS.SYMBOL = ARRAY(NTRANS);  FILL(.TRANS.SYMBOL,TS)
         TRANS.ACTION = ARRAY(NTRANS);  FILL(.TRANS.ACTION,TA)
         FIRST.TRANS = ARRAY(NSTATES);  FILL(.FIRST.TRANS,FT)
         LAST.TRANS  = ARRAY(NSTATES);  FILL(.LAST.TRANS,LT)
         LHS.SYMBOL  = ARRAY(NPRODS);   FILL(.LHS.SYMBOL,LS)
         RHS.LENGTH  = ARRAY(NPRODS);   FILL(.RHS.LENGTH,RL)
+                                                     :(READ)
*
FILL     I = I + 1
         STRING  SPAN('0123456789-') . VAL ',' = :F(RETURN)
         ITEM($VECTOR,I) = CONVERT(VAL,.INTEGER) :(FILL)
*
*
READ     CARD = INPUT                                 :F(END)
RESUME   CARD @P KEYWORD                              :S(PREPARE)
         OUTPUT = DIFFER(TRIM(CARD))  CARD            :(READ)
*
PREPARE CARD  LEN(P) . SNARK =
         OUTPUT = DIFFER(TRIM(SNARK)) ' ' SNARK  :($KEY)
*
*  INITIALIZE  --  COPY IN SYSTEM DECLARES AND SUBROUTINES
*                  DETERMINE ALSO ROUNDING MODE.
*
INITIALIZE    SCAN()
         INITFILE = 'SNORT.RND'
         INITFILE = IDENT(SCAN(),'CHOPPED') 'SNORT.CHP'
         INPUT(.INIT,INITFILE)
INITCOPY  OUTPUT = INIT                               :S(INITCOPY)
         SCAN()
         OUTPUT = NE(SCANTYPE,SEMICOLON) '  /***** ERROR IN INITIALIZE'
+                ' STATEMENT *****/'         :(RESUME)
*
*  SMEAR --  GENERATE DECLARATIONS FOR SMEARED VARIABLES
*
SMEAR    SCAN()
         OUTPUT = '  DECLARE'
SMEARLOOP     V = SCAN()
         EQ(SCANTYPE,VARIABLE)                        :F(SMERROR)
         SCAN()
         EQ(SCANTYPE,COMMA)                           :F(SMEARFIN)
         OUTPUT = '      1 ' V ' LIKE SMEARED_NUMBER,'  :(SMEARLOOP)
SMEARFIN  EQ(SCANTYPE,SEMICOLON)                      :F(SMERROR)
SMEND    OUTPUT = '      1 ' V ' LIKE SMEARED_NUMBER;'  :(RESUME)
SMERROR OUTPUT = '  /***** ERROR IN SMEAR LIST. END FORCED *****/'
                                                      :(RESUME)
*
*  PRINT  --  FORMATTED PRINT OF SMEARED VARIABLES
*
PRINT    SCAN()
PRLIST   V = SCAN()
         EQ(SCANTYPE,VARIABLE)                        :F(PRERROR)
         OUTPUT = "  CALL SMEARED_PRINT('" V "'," V ");"
         SCAN()
         EQ(SCANTYPE,COMMA)                           :S(PRLIST)
         EQ(SCANTYPE,SEMICOLON)                       :S(RESUME)
```

```
PRERROR OUTPUT = '  /***** ERROR IN PRINT LIST.  END FORCED *****/'
                                                         :(RESUME)
-EJECT
*
*  EVAL    --  PARSE EXPRESSIONS OF SMEARED NUMBERS AND
*               GENERATE APPROPRIATE SUBROUTINE CALLS.
*
EVAL    SCAN()
        PARSE.EXPR()                                     :(RESUME)
*
SEMANTICS                                                :($('PROD' PRODNUM))
*-----------------------------------------------------------------------*
*  1:            S   ==>  <ASSIGNMENT> ;                                *
*-----------------------------------------------------------------------*
PROD1                                                    :(RETURN)
*-----------------------------------------------------------------------*
*  2:            <ASSIGNMENT>  ==>  <VAR>  =  <EXPR>                    *
*-----------------------------------------------------------------------*
PROD2   LASTEMIT(TOKENSTACK<STACKPTR>)
        CLEARREGS()                                      :(RETURN)
*-----------------------------------------------------------------------*
*  3:            <EXPR>  ==>  <EXPR>  +  <TERM>                         *
*-----------------------------------------------------------------------*
PROD3   FREE.REGISTER(TOKENSTACK<STACKPTR>)
        FREE.REGISTER(TOKENSTACK<STACKPTR + 2>)
        AREG = ALLOCATED.REG()
        EMIT( '  CALL SMEARED_ADD(' AREG ',' TOKENSTACK<STACKPTR>
+              ',' TOKENSTACK<STACKPTR + 2> ');' )
        TOKENSTACK<STACKPTR> = AREG                      :(RETURN)
*-----------------------------------------------------------------------*
*  4:            <EXPR>  ==>  <EXPR>  -  <TERM>                         *
*-----------------------------------------------------------------------*
PROD4   FREE.REGISTER(TOKENSTACK<STACKPTR>)
        FREE.REGISTER(TOKENSTACK<STACKPTR + 2>)
        AREG = ALLOCATED.REG()
        EMIT( '  CALL SMEARED_SUB(' AREG ',' TOKENSTACK<STACKPTR>
+              ',' TOKENSTACK<STACKPTR + 2> ');' )
        TOKENSTACK<STACKPTR> = AREG                      :(RETURN)
*-----------------------------------------------------------------------*
*  5:            <EXPR>  ==>  <TERM>                                    *
*-----------------------------------------------------------------------*
PROD5                                                    :(RETURN)
*-----------------------------------------------------------------------*
*  6:            <EXPR>  ==>  +  <TERM>                                 *
*-----------------------------------------------------------------------*
PROD6   TOKENSTACK<STACKPTR> = TOKENSTACK<STACKPTR + 1>  :(RETURN)
*-----------------------------------------------------------------------*
*  7:            <EXPR>  ==>  -  <TERM>                                 *
*-----------------------------------------------------------------------*
PROD7   FREE.REGISTER(TOKENSTACK<STACKPTR + 1>)
        AREG = ALLOCATED.REG()
        EMIT( '  CALL SMEARED_NEGATE('  AREG  ','
+                TOKENSTACK<STACKPTR + 1> ');' )
        TOKENSTACK<STACKPTR> = AREG                      :(RETURN)
*-----------------------------------------------------------------------*
*  8:            <TERM>  ==>  <TERM>  *  <FACTOR>                       *
*-----------------------------------------------------------------------*
PROD8   FREE.REGISTER(TOKENSTACK<STACKPTR>)
        FREE.REGISTER(TOKENSTACK<STACKPTR + 2>)
        AREG = ALLOCATED.REG()
```

```
        EMIT( '  CALL SMEARED_MULT(' AREG ',' TOKENSTACK<STACKPTR>
+            ',' TOKENSTACK<STACKPTR + 2> ');' )
        TOKENSTACK<STACKPTR> = AREG                    :(RETURN)
*-----------------------------------------------------------------*
*  9:           <TERM>  ==>  <TERM>  /  <FACTOR>                  *
*-----------------------------------------------------------------*
PROD9   FREE.REGISTER(TOKENSTACK<STACKPTR>)
        FREE.REGISTER(TOKENSTACK<STACKPTR + 2>)
        AREG = ALLOCATED.REG()
        EMIT( '  CALL SMEARED_DIV(' AREG ',' TOKENSTACK<STACKPTR>
+            ',' TOKENSTACK<STACKPTR + 2> ');' )
        TOKENSTACK<STACKPTR> = AREG                    :(RETURN)
*-----------------------------------------------------------------*
*  10:          <TERM>  ==>  <FACTOR>                             *
*-----------------------------------------------------------------*
PROD10                                                 :(RETURN)
*-----------------------------------------------------------------*
*  11:          <FACTOR>  ==>  (  <EXPR>  )                       *
*-----------------------------------------------------------------*
PROD11  TOKENSTACK<STACKPTR> = TOKENSTACK<STACKPTR + 1>  :(RETURN)
*-----------------------------------------------------------------*
*               <FACTOR>  ==>  <VAR>                              *
*-----------------------------------------------------------------*
PROD12  EQ(VARINFO,0)                                  :S(RETURN)
CONSTANT.VAR  AREG = ALLOCATED.REG()
        EMIT( '  CALL SETCONSTANT('  AREG  ','
+            TOKENSTACK<STACKPTR>  ');' )
        TOKENSTACK<STACKPTR> = AREG                    :(RETURN)
*-----------------------------------------------------------------*
ALLOCATED.REG   I = LT(I,NREGS)  I + 1                 :F(ALL.USED)
        REGISTER<I> = IDENT(REGISTER<I>) 'X'           :F(ALLOCATED.REG)
ALLOCATE  ALLOCATED.REG = 'REG(' I ')'                 :(RETURN)
ALL.USED  OUTPUT = '  /***** ALL REGISTERS USED *****/' :(ALLOCATE)
*
FREE.REGISTER    REGNAME  'REG('   BREAK(')') . I    :F(RETURN)
        REGISTER<I> =                                  :(RETURN)
*
CLEARREGS       I = LT(I,NREGS) I + 1                  :F(RETURN)
                REGISTER<I> =                          :(CLEARREGS)
*
EMIT    OUTPUT = DIFFER(SAVED.STRING) SAVED.STRING
        SAVED.STRING = STRING                          :(RETURN)
*
LASTEMIT  SAVED.STRING  (BREAK('(') LEN(1)) . X  BREAK(',') =
+                       X     TARGET                   :F(ASSMT)
        OUTPUT = SAVED.STRING
        SAVED.STRING =                                 :(RETURN)
ASSMT   OUTPUT = '   ' TARGET ' = ' TOKENSTACK<STACKPTR + 2> ';'
+                                                      :(RETURN)
-EJECT
*-----------------------------------------------------------------*
*
*               SLR(1) EXPRESSION PARSER
*
*-----------------------------------------------------------------*
PARSE.EXPR      STATESTACK<1> = 1
                SYMBOLSTACK<1> = 0
                CURRENT.STATE = 1
                STACKPTR = 2
                NEEDREAD = 'YES'
```

```
*
PARSE    I = FIRST.TRANS<CURRENT.STATE>
         J =  LAST.TRANS<CURRENT.STATE>
         EQ(I,0)                                  :S(REDUCE.LR0)
         NEEDREAD = IDENT(NEEDREAD) 'YES'         :S(FIND)
         LOOKAHEAD.TOKEN = SCAN()
         LOOKAHEAD.SYMBOL = SCANTYPE
FIND     EQ(LOOKAHEAD.SYMBOL,TRANS.SYMBOL<I>)     :S(FOUND)
         I = LT(I,J) I + 1                        :S(FIND)F(SYNTAXERR)
FOUND    J = TRANS.ACTION<I>
         EQ(J,0)                                  :S(ACCEPT)
         LT(J,0)                                  :S(REDUCE.LR1)
*
SHIFT    STATESTACK<STACKPTR> = J
         SYMBOLSTACK<STACKPTR> = LOOKAHEAD.SYMBOL
         TOKENSTACK<STACKPTR> = LOOKAHEAD.TOKEN
         CURRENT.STATE = J
         STACKPTR = LT(STACKPTR,DEPTH) STACKPTR + 1  :S(PARSE)
         OUTPUT = '    /******* STACK OVERFLOW ******/' :(PARSE)
*
REDUCE.LR0       NEEDREAD = 'YES'                 :(REDUCE)
REDUCE.LR1       NEEDREAD =
REDUCE   PROD = -J
         STACKPTR = STACKPTR - RHS.LENGTH<PROD>
         CURRENT.STATE = STATESTACK<STACKPTR - 1>
         SYMBOLSTACK<STACKPTR> = LHS.SYMBOL<PROD>
         I = FIRST.TRANS<CURRENT.STATE>
         J =  LAST.TRANS<CURRENT.STATE>
GOTO     EQ(SYMBOLSTACK<STACKPTR>,TRANS.SYMBOL<I>)      :S(PUSHGOTO)
         I = LT(I,J) I + 1                        :S(GOTO)F(SYNTAXERR)
PUSHGOTO    J = TRANS.ACTION<I>
         STATESTACK<STACKPTR> = J
         CURRENT.STATE = J
         SEMANTICS(PROD)
         STACKPTR = STACKPTR + 1                  :(PARSE)
*
ACCEPT                                            :(RETURN)
SYNTAXERR OUTPUT = '  /***** SYNTAX ERROR *****/'  :(FRETURN)
*
*-----------------------------------------------------------------*
-EJECT
*
*        SCAN -- LEXICAL ANALYZER.    RETURNED VALUE IS TOKEN STRING.
*                ALSO RETURNS   SCANTYPE -- NUMERICAL TOKEN CODE
*                               VARINFO -- EXTRA INFO FOR <VAR> TOKENS
*
SCAN     SCAN =
         NEXTCHAR =
         GETNONBLANK()                  :($('CLASS' CLASSN))
*-----------------------------------------------------------------*
*        CLASS 1 -- LETTERS, BREAK CHARACTERS:  BUILD VARIABLES
*-----------------------------------------------------------------*
CLASS1   SCANTYPE = VARIABLE;  VARINFO = 0
CLASS1A  ADD();  GETCHAR()
         GE(CLASSN,1) LE(CLASSN,2)                :S(CLASS1A)
CLASS1B  IDENT(NEXTCHAR,' ')                      :F(CLASS1C)
         GETNONBLANK()
CLASS1C  EQ(CLASSN,8)                             :F(OUT)
         PARENCOUNT = 1
CLASS1D  ADD();  GETCHAR()
```

```
        PARENCOUNT = EQ(CLASSN,8) PARENCOUNT + 1 :S(CLASS10)
        PARENCOUNT = EQ(CLASSN,9) PARENCOUNT - 1 :F(CLASS10)
        EQ(PARENCOUNT,0)                         :F(CLASS10)
        ADD()                                    :(OUT)
*-----------------------------------------------------------------*
*       CLASS 2 -- DIGITS:  BUILD CONSTANTS
*-----------------------------------------------------------------*
CLASS2  SCANTYPE = VARIABLE;  VARINFO = 1
CLASS2A ADD();  GETNONBLANK()
        VARINFO = VARINFO + 1
        EQ(CLASSN,2)                           :S(CLASS2A)
        VARINFO = IDENT(NEXTCHAR,'.') 100      :S(CLASS2A)
        IDENT(NEXTCHAR,'E')                    :F(OUT)
        ADD();  GETCHAR()
        GE(CLASSN,2) LE(CLASSN,4)              :F(OUT)
CLASS2B ADD();  GETCHAR()
        EQ(CLASSN,2)                           :S(CLASS2B)F(OUT)
*-----------------------------------------------------------------*
*       CLASSES 3-10 -- TERMINALS    3+ 4- 5/ 6* 7= 8( 9) 10;
*-----------------------------------------------------------------*
CLASS3  ADD();  GETCHAR();  SCANTYPE = PLUS
        DIFFER(UNARY) EQ(CLASSN,2)             :S(CLASS2)F(OUT)
CLASS4  ADD();  GETCHAR();  SCANTYPE = MINUS
        DIFFER(UNARY) EQ(CLASSN,2)             :S(CLASS2)F(OUT)
CLASS5  ADD();  GETCHAR();  SCANTYPE = SLASH
        EQ(CLASSN,6)                           :F(OUT)
CLASS5A REMOVE();  GETNONBLANK()
CLASS5B EQ(CLASSN,6)                           :F(CLASS5A)
        REMOVE();  GETNONBLANK()
        EQ(CLASSN,5)                           :F(CLASS5B)
        REMOVE() .                             :(SCAN)
CLASS6  ADD();  SCANTYPE = STAR                :(OUT)
CLASS7  ADD();  SCANTYPE = EQUALS              :(OUT)
CLASS8  ADD();  SCANTYPE = LPAREN              :(OUT)
CLASS9  ADD();  SCANTYPE = RPAREN              :(OUT)
CLASS10 ADD(); SCANTYPE = SEMICOLON            :(OUT)
CLASS11 ADD(); SCANTYPE = COMMA                :(OUT)
CLASS12 IMPOSSIBLE
CLASS0  REMOVE()
        OUTPUT = "  /****** ERROR, UNKNOWN CHARACTER '"
+          NEXTCHAR "' ENCOUNTERED:  DELETED. ******/"  :(SCAN)
*
OUT     UNARY =
        UNARY = EQ(SCANTYPE,EQUALS) '1'
        UNARY = EQ(SCANTYPE,LPAREN) '1'
+                                              :(RETURN)
*-----------------------------------------------------------------*
GETCHAR CARD LEN(1) . NEXTCHAR                 :F(MORE)
        CLASSN = IDENT(NEXTCHAR,' ') 12        :S(RETURN)
        CLASSN = IDENT(NEXTCHAR,'        ') 12 :S(RETURN)
        CLASSN = IDENT(NEXTCHAR,',') 11        :S(RETURN)
        CLASSN = REPLACE(NEXTCHAR,
+           'ABCDEFGHIJKLMNOPQRSTUVWXYZ@$_#0123456789+-/*=();',
+           '000000000000000000000000000000111111111123456789')
        CLASSN = INTEGER(CLASSN) CONVERT(CLASSN,.INTEGER) + 1
+                                              :S(RETURN)
        CLASSN = 0                             :(RETURN)
MORE    CARD = INPUT                           :S(GETCHAR)
        OUTPUT = ' /****** EOF HIT *******/'
        CLASSN = 10
```

```
          NEXTCHAR = ';'                               :(RETURN)
*
GETNONBLANK        GETCHAR()
                   EQ(CLASSN,12)                       :F(RETURN)
                   CARD LEN(1) =                       :(GETNONBLANK)
*
ADD       SCAN = SCAN NEXTCHAR
          CARD LEN(1) =                                :(RETURN)
*
REMOVE    CARD LEN(1) =                                :(RETURN)
-EJECT
END
```

SNORTRAN System Routines

```
        DECLARE
            1     SMEARED_NUMBER,
              2   COMPUTED_VALUE   FLOAT BINARY(21),
              2   VARIANCE         FLOAT BINARY(21), /*UNITS BETA**(-2T)*/
              2   INTDATA,
                  3   IA_LB        FLOAT BINARY(21), /* INTERVAL            */
                  3   IA_UB        FLOAT BINARY(21); /*  ANALYSIS DATA      */

        DECLARE
                  1 REG(10) LIKE SMEARED_NUMBER,
                  BETA_MT   FLOAT BINARY(21) INIT(1.0E-248), /*16**(-6)*/
                  HUGE      FLOAT BINARY(21) INIT(1.0E+888);

    SMEARED_ADD:  PROC (A,B,C);
        /*  PERFORMS ADDITION  A = B + C  FOR SMEARED NUMBERS  */
        DECLARE 1 (A,B,C) LIKE SMEARED_NUMBER,
                  (T,T1,T2,T3) FLOAT BINARY(53);
        T = B.COMPUTED_VALUE;
        A.COMPUTED_VALUE = ROUND(T + C.COMPUTED_VALUE);
        T2 = MAXSQ(B.INTDATA);
        T3 = MAXSQ(C.INTDATA);
        CALL INTERVAL(A.INTDATA,1 /*ADD*/, B.INTDATA,C.INTDATA);
        T1 = MINSQ(A.INTDATA);
        IF ((T2+T3)/T1) > HUGE  THEN
                DO;  A.VARIANCE = HUGE; RETURN;  END;
        A.VARIANCE = ROUNDUP( T2/T1*B.VARIANCE + T3/T1*C.VARIANCE );
        CALL ADD_ROUNDING_ERROR(A);
        END SMEARED_ADD;

    SMEARED_NEGATE: PROC (A,B);
        /*  PERFORMS NEGATION  A = -B  FOR SMEARED NUMBERS  */
        DECLARE  1 (A,B) LIKE SMEARED_NUMBER,
                   T FLOAT BINARY(21);
        A.COMPUTED_VALUE = -B.COMPUTED_VALUE;
        A.VARIANCE = B.VARIANCE;
        T = B.IA_LB;
        A.IA_LB = -B.IA_UB;
        A.IA_UB = -T;
        END SMEARED_NEGATE;

    SMEARED_SUB:  PROC (A,B,C);
        /*  PERFORMS SUBTRACTION  A = B - C  FOR SMEARED NUMBERS  */
        DECLARE 1 (A,B,C) LIKE SMEARED_NUMBER,
                  (T,T1,T2,T3) FLOAT BINARY(53);
        T = B.COMPUTED_VALUE;
        A.COMPUTED_VALUE = ROUND(T - C.COMPUTED_VALUE);
        T2 = MAXSQ(B.INTDATA);
        T3 = MAXSQ(C.INTDATA);
        CALL INTERVAL(A.INTDATA,2 /*SUB*/, B.INTDATA,C.INTDATA);
        T1 = MINSQ(A.INTDATA);
        IF ((T2+T3)/T1) > HUGE  THEN
                DO;  A.VARIANCE = HUGE; RETURN;  END;
        A.VARIANCE = ROUNDUP( T2/T1*B.VARIANCE + T3/T1*C.VARIANCE );
        CALL ADD_ROUNDING_ERROR(A);
        END SMEARED_SUB;

    SMEARED_MULT:  PROC(A,B,C);
        /*  PERFORMS MULTIPLICATION  A = B * C  FOR SMEARED NUMBERS  */
        DECLARE  1  (A,B,C) LIKE SMEARED_NUMBER,
```

```
                   T FLOAT BINARY(53);
            T = B.COMPUTED_VALUE;
            A.COMPUTED_VALUE = ROUND(T * C.COMPUTED_VALUE);
            CALL INTERVAL(A.INTDATA,3 /*MULT*/,B.INTDATA,C.INTDATA);
            T = B.VARIANCE;
            A.VARIANCE = ROUNDUP(T + C.VARIANCE);
            CALL ADD_ROUNDING_ERROR(A);
            END SMEARED_MULT;

    SMEARED_DIV:  PROC(A,B,C);
            /*  PERFORMS DIVISION  A = B / C  FOR SMEARED NUMBERS  */
            DECLARE  1  (A,B,C) LIKE SMEARED_NUMBER,
                   T FLOAT BINARY(53);
            T = B.COMPUTED_VALUE;
            A.COMPUTED_VALUE = ROUND(T / C.COMPUTED_VALUE);
            CALL INTERVAL(A.INTDATA,4 /*DIV*/,B.INTDATA,C.INTDATA);
            T = B.VARIANCE;
            A.VARIANCE = ROUNDUP(T + C.VARIANCE);
            CALL ADD_ROUNDING_ERROR(A);
            END SMEARED_DIV;

    SETCONSTANT: PROC(A,C);
            /*  SETS SMEARED NUMBER  A  TO THE INITIAL VALUE  C  */
            DECLARE  1  A  LIKE SMEARED_NUMBER,
                   C FLOAT BINARY(53);
            A.COMPUTED_VALUE = ROUND(C);
            A.IA_LB = ROUNDDOWN(C);
            A.IA_UB = ROUNDUP(C);
            A.VARIANCE = 0E00;
            IF C ¬= PRECISION(C,21) THEN
                  CALL ADD_ROUNDING_ERROR(A);
            END SETCONSTANT;

    INTERVAL:  PROC(A,OP,B,C);
            /*  PERFORMS INTERVAL ARITHMETIC   A = B OP C   */
            DECLARE 1 INTERVAL,
                  2 IA_LB      FLOAT BINARY(21),
                  2 IA_UB      FLOAT BINARY(21),
                1 (A,B,C) LIKE INTERVAL,
                (S1,S2,S3,S4) FLOAT BINARY(53),
                IOP(4) LABEL,
                OP      FIXED BINARY(15);
            S1 = B.IA_LB; S2 = B.IA_UB; S3 = C.IA_LB; S4 = C.IA_UB;
            GO TO IOP(OP);

            IOP(1):      /*  ADDITION  */
            A.IA_LB = ROUNDDOWN(S1 + S3);
            A.IA_UB = ROUNDUP  (S2 + S4);
            RETURN;

            IOP(2):    /*  SUBTRACTION  */
            A.IA_LB = ROUNDDOWN(S1 - S4);
            A.IA_UB = ROUNDUP  (S2 - S3);
            RETURN;

            IOP(3):    /*  MULTIPLICATION  */
            A.IA_LB = MIN(    ROUNDDOWN(S1 * S3),
                              ROUNDDOWN(S1 * S4),
                              ROUNDDOWN(S2 * S3),
                              ROUNDDOWN(S2 * S4)    );
```

```
        A.IA_UB = MAX(   ROUNDUP   (S1 * S3),
                         ROUNDUP   (S1 * S4),
                         ROUNDUP   (S2 * S3),
                         ROUNDUP   (S2 * S4)    );
        RETURN;

        IOP(4):    /*  DIVISION  */
        A.IA_LB = MIN(   ROUNDDOWN(S1 / S3),
                         ROUNDDOWN(S1 / S4),
                         ROUNDDOWN(S2 / S3),
                         ROUNDDOWN(S2 / S4)    );
        A.IA_UB = MAX(   ROUNDUP   (S1 / S3),
                         ROUNDUP   (S1 / S4),
                         ROUNDUP   (S2 / S3),
                         ROUNDUP   (S2 / S4)    );
        RETURN;

        END INTERVAL;

MAXSQ:  PROC(A)  RETURNS(FLOAT BINARY(53));
        /*  RETURNS MAX VALUE OF A**2 OVER INTERVAL A  */
        DECLARE 1 A,  /* INTERVAL  */
                2 LB FLOAT BINARY(21),
                2 UB FLOAT BINARY(21),
                  (T1,T2) FLOAT BINARY(53);
        T1 = PRECISION(A.LB,53)**2;
        T2 = PRECISION(A.UB,53)**2;
        T1 = MAX(T1,T2);
        RETURN(T1);
        END MAXSQ;

MINSQ:  PROC(A)  RETURNS(FLOAT BINARY(53));
        /*  RETURNS MIN VALUE OF A**2 OVER INTERVAL A  */
        DECLARE 1 A,  /* INTERVAL  */
                2 LB FLOAT BINARY(21),
                2 UB FLOAT BINARY(21),
                  (T1,T2) FLOAT BINARY(53);
        IF (A.LB*A.UB <= 0E0B) THEN   /*  INTERVAL CONTAINS ZERO  */
                DO;  T1 = 1E0B/HUGE;  RETURN(T1); END;
        T1 = PRECISION(A.LB,53)**2;
        T2 = PRECISION(A.UB,53)**2;
        T1 = MIN(T1,T2);
        RETURN(T1);
        END MINSQ;


ADD_ROUNDING_ERROR: PROC(A);
        DECLARE 1 A LIKE SMEARED_NUMBER,
                (D,S) FLOAT BINARY(53);
        /*  COMPUTE        D = MIN(MANTISSA(A))                    */
        /*                 S =  1 / (2D)**2  / 3                   */
        /*                   = COMPUTATION ROUNDOFF ERROR VARIANCE, */
        /*                      UNITS BETA**(-2T).                 */

        IF A.VARIANCE >= HUGE | A.COMPUTED_VALUE=0E0B THEN RETURN;
        D = MIN(MANTISSA(A.IA_LB),MANTISSA(A.IA_UB));
        S = 1E0B / (D*D*12E0);
        A.VARIANCE = ROUNDUP(A.VARIANCE + D);
        END ADD_ROUNDING_ERROR;
```

```
   MANTISSA:  PROC(X)  RETURNS(FLOAT BINARY(53));
          DECLARE (X,Y) FLOAT BINARY(21);
          Y = ABS(X);
                 DO WHILE (Y > 1.0000000000000000000000E0B);
                 Y = Y * 1.0000000000000000000000E-4B;
                 END;
                 DO WHILE (Y < 1.0000000000000000000000E-4B);
                 Y = Y * 1.0000000000000000000000E+4B;
                 END;
          RETURN(Y);
          END MANTISSA;

   SMEARED_PRINT:  PROC(AA,A);
          DECLARE 1 A LIKE SMEARED_NUMBER,
                 AA  CHAR(*),
                 (S,S1,S2,ABS1,ABS2,ABS3,ABS4) FLOAT BIN(53);
          S = SQRT(A.VARIANCE);
          S1 = -3.625 * S;    /* 99.7% CONFIDENCE LIMITS  */
          S2 = +3.625 * S;
          ABS3 = A.COMPUTED_VALUE * (1 + S1*BETA_MT);
          ABS4 = A.COMPUTED_VALUE * (1 + S2*BETA_MT);
          IF A.COMPUTED_VALUE < 0  THEN
                 DO;  ABS1=ABS3; ABS3=ABS4; ABS4=ABS1;  END;

          PUT FILE(SYSPRINT) SKIP EDIT

          (AA,'COMPUTED VALUE:',A.COMPUTED_VALUE)
                 (SKIP,COL(7),A,X(8),A,E(25,15,16))

          ('3.625*SIGMA BOUNDS ON RELATIVE ERROR:','(',S1,'  ,',S2,
            ' ) * BETA**(-T)' )
                 (SKIP,COL(12),A,COL(50),A,2 (E(25,15,16),A))

          ('IMPLIED 99.7%-CONFIDENCE BOUNDS:', '(', ABS3, '  ,',
                 ABS4, ' )'  )
                 (SKIP,COL(12),A,COL(50),A,2 (E(25,15,16),A))

          ('INTERVAL ANALYSIS BOUNDS:', '(',A.IA_LB,'  ,', A.IA_UB,' )')
                 (SKIP,COL(12),A,COL(50),A,2 (E(16,6,7),X(9),A));

          END SMEARED_PRINT;


  ROUND: PROC(X) RETURNS(FLOAT BINARY(21));
          DECLARE (X,HALF) FLOAT BINARY(53),
                 RX         FLOAT BINARY(21),
                 E          BIT(64),
                 EEXPT      BIT(8) DEFINED E POSITION(1),
                 RBIT       BIT(1) DEFINED E POSITION(33),
                 F          BIT(64) INIT(
   '0000000000000000000000000000000010000000000000000000000000000000'B),
                 FEXPT      BIT(8) DEFINED F POSITION(1);
          E = UNSPEC(X);
          IF  RBIT  THEN  DO;      /*  ROUND  */
                 FEXPT = EEXPT;
                 UNSPEC(HALF) = F;
                 RX = X + HALF;
                 END;
             ELSE   RX = X;        /*  TRUNCATE  */
```

```
         RETURN(RX);
         END ROUND;

ROUNDUP: PROC(X) RETURNS(FLOAT BINARY(21));
         DECLARE X FLOAT BINARY(53),
                 RX FLOAT BINARY(21);
         IF X > OEOB  THEN RETURN(ROUND(X));
                      ELSE DO; RX=X; RETURN(RX); END;
         END ROUNDUP;

ROUNDDOWN: PROC(X) RETURNS(FLOAT BINARY(21));
         DECLARE X FLOAT BINARY(53),
                 RX FLOAT BINARY(21);
         IF X < OEOB  THEN RETURN(ROUND(X));
                      ELSE DO; RX=X; RETURN(RX); END;
         END ROUNDDOWN;
```

7. CONCLUSIONS

The important conclusions of this paper are first, that accumulated roundoff errors tend to cluster towards the center of their bounds like normal densities; second, that high-confidence bounds can be proved; third, that automated statistical analysis is not worthwhile.

The last conclusion is not obvious. Naturally the statistical model will generate better bounds than the worst-case model of Wilkinson. But beyond that no conclusions are obvious. Interval Analysis treats error densities as uniform, although the Central Limit Theorem tells us they are almost normal. We would expect better results from statistical analysis because it not only combines intervals (read "probability densities on intervals"), it also retains information about the resultant error distribution on the new interval. But statistical analysis breaks down because relative errors cannot handle cancellation or numbers near zero, and correlation is almost impossible to monitor practically.

One possible exit from the irrepresentability of zero is to drop relative errors altogether and develop a statistical theory of <u>absolute</u> errors. Observations 2 and 3 of Section 2 provide most of the necessary theory if we now treat numbers as random variables instead of the number's relative errors. Zero then becomes representable. However, it is not clear that an analogue of Theorem 5 in Section 4 can be proved for absolute analysis since multiplication and division no longer involve simple convolution, and besides, the problems of correlated errors cannot be eliminated. Statistical error analysis in general seems of questionable use.

REFERENCES

1. Feller, W. An Introduction to Probability Theory and Its Applications v. II. NY: John Wiley & Sons, 1966.

2. Gnedenko, B. V. and A. N. Kolmogorov. Limit Distributions for Sums of Independent Random Variables. Reading, Mass.: Addison-Wesley, 1954.

3. Hamming, R. W. "On the Distribution of Numbers" Bell Sys. Tech. J. 49, 8, 1609 (1970).

4. Henrici, P. Elements of Numerical Analysis. NY: John Wiley & Sons, 1964.

5. Huskey, H.D. "On the Precision of a Certain Procedure of Numerical Integration." J. Res. Nat. Bur. Stds., 42, 1, 57 (1949).

6. Kaneko, T. and B. Liu. "On Local Roundoff Errors in Floating-Point Arithmetic." JACM 20, 3, 391 (1973).

7. Lever Brothers. "Vanish Advertisement." Shopper's Weekly Magazine, 17 May 1975.

8. McBride, E. B. Obtaining Generating Functions. NY: Springer-Verlag, 1971.

9. Marasa, J. D. and D. W. Matula. "A Simulative Study of Correlated Error Propagation in Various Finite-Precision Arithmetics." IEEE Trans Comp. C-22, 6, 587 (1973).

10. Papoulis, A. The Fourier Integral and its Applications. NY: McGraw-Hill, 1962.

11. Pennington, R. H. Introductory Computer Methods and Numerical Analysis. Toronto: Macmillan, 1970.

12. Prohorov, Yu. V. and Yu. A. Rozanov. Probability Theory. NY: Springer-Verlag, 1969.

13. Sterbenz, P. H. Floating-Point Computation. Englewood Cliffs, NJ: Prentice-Hall, 1974.

14. Tsao, N. "On the Distribution of Significant Digits and Roundoff Errors." CACM 17, 5, 269 (1974).

15. Uspensky, J. V. Introduction to Mathematical Probability. NY: McGraw-Hill, 1937.

16. Wilkinson, J. H. Rounding Errors in Algebraic Processes. Englewood Cliffs, NJ: Prentice-Hall, 1963.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-76-787	2.	3. Recipient's Accession No.

4. Title and Subtitle

THE STATISTICAL THEORY OF RELATIVE ERRORS IN FLOATING-POINT COMPUTATION

5. Report Date: March 1976

6.

7. Author(s)

Douglass Stott Parker, Jr.

8. Performing Organization Rept. No. UIUCDCS-R-76-787

9. Performing Organization Name and Address

University of Illinois at Urbana-Champaign
Department of Computer Science
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

US NSF DCR73-07980 A02

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D. C.

13. Type of Report & Period Covered

Master's Thesis

14.

15. Supplementary Notes

16. Abstracts

Experience with interval analysis indicates that the error bounds it generates for most floating-point computations are pessimistic. This is necessary since interval analysis assumes some error in every computation, and is oblivious of both error cancellation and the Central Limit Theorem. It has been questioned whether a possible alternative to this generation of bounds is the use of error probability densities to obtain high-confidence intervals, or statistical bounds, on the computed error. This paper shows that such a probabilistic system for Wilkinsonian relative errors can be derived, but points out numerous crippling drawbacks for this system and statistical error analysis in general.

17. Key Words and Document Analysis. 17a. Descriptors

Automated error analysis
Floating-point computation
Interval analysis
Roundoff error
Statistical error bounds

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release Unlimited	UNCLASSIFIED	66
	20. Security Class (This Page) UNCLASSIFIED	22. Price